Python 数据分析

基于Plotly的动态可视化绘图

孙洋洋　王硕　邢梦来　袁泉　吴娜　著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

VISIT…

内 容 简 介

随着信息技术的发展和硬件设备成本的降低，当今的互联网存在海量的数据，要想快速从这些数据中获取更多有效的信息，数据可视化是重要的一环。对于 Python 语言来说，比较传统的数据可视化模块是 Matplotlib，但它存在不够美观、静态性、不易分享等缺点，限制了 Python 在数据可视化方面的发展。为了解决这个问题，新型的动态可视化开源模块 Plotly 应运而生。由于 Plotly 具有动态、美观、易用、种类丰富等特性，所以一经问世就受到开发人员的喜爱。本书主要介绍 Plotly 在可视化各领域的应用，包括基础绘图、数据处理、网页开发、程序 GUI、机器学习和量化投资等，方便读者对 Plotly 快速上手。

本书绝大部分代码用 Python 语言编写，同时也给出了 Plotly 在 R 语言、MATLAB 和 JavaScript 中的应用案例。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

Python 数据分析：基于 Plotly 的动态可视化绘图 / 孙洋洋等著. —北京：电子工业出版社，2018.6
ISBN 978-7-121-34113-7

Ⅰ. ①P… Ⅱ. ①孙… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2018）第 083114 号

策划编辑：黄爱萍
责任编辑：牛 勇
印 刷：三河市华成印务有限公司
装 订：三河市华成印务有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编：100036
开 本：787×980 1/16 印张：25 字数：460 千字
版 次：2018 年 6 月第 1 版
印 次：2018 年 6 月第 1 次印刷
定 价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。

前　言

Python 是一门非常优秀的编程语言，其语法简洁、易学易用，越来越受到编程人员的喜爱；Python 也是一门非常“人性化”的编程语言，其各种语法规则的设计符合人们的思维方式，开发人员可以用最简单的方式实现自己的编程目的，降低时间成本；同时，Python 又是一门非常强大的编程语言，其在编程的各个领域都有非常不错的表现，比如在网页开发、程序 GUI 设计、网络爬虫、科学计算、数据可视化、机器学习与深度学习等领域，Python 都有非常好的解决方案来解决现实中的业务问题。

互联网的快速发展为我们积累了庞大的数据，计算机硬件的创新为存储与分析这些数据创造了硬件条件，编程语言的发展为分析这些数据创造了软件条件。在数据分析这个领域，Python 有着自己独有的优势，简单易用的特性与强大的开源模块的支持使其成为数据分析领域方便、好用的利器。

Python 在数据分析领域的广泛应用离不开其强大的开源模块的支持，大名鼎鼎的 NumPy、SciPy、StatsModels、Pandas 等模块的建立与发展奠定了 Python 在数据分析领域中的重要地位。这些模块简单又好用，它们提供的解决方案能够满足绝大部分业务需求。在人工智能领域，Python 也有非常棒的解决方案，如 Sklearn、TensorFlow、MXNet、Theano、PyTorch、Caffe 等都是非常好的开源模块。尤其在人工智能中最前沿的深度学习领域，Python 几乎占据了霸主的地位。Python 借助在数据分析领域中开源模块的优势，在量化投资领域逐渐占据了领头羊的地位。国内外主流量化投资网站大多支持 Python 语言，其在量化投资领域有一种逐渐淘汰其他语言，一统“江湖”之势。

对数据的分析离不开数据的可视化，相对于 Python 在数据分析、人工智能、量化投资等领域中的发展，在数据可视化方面的发展有些滞后。最经典的 Python 可视

化绘图库莫过于 Matplotlib 了，Matplotlib 就是 MATLAB+Plot+Library，即模仿 MATLAB 的绘图库，其绘图风格与 MATLAB 类似。由于 MATLAB 的绘图风格有些偏古典，为了绘出更漂亮的图像，Python 开源社区开发出了 Seaborn 绘图模块，它本质上是对 Matplotlib 的封装，绘图效果更符合现代人的审美观。尽管如此，由于 Matplotlib 是基于 GUI 的绘图模块，所以存在特有的缺陷。

就笔者使用的经验而言，Matplotlib 主要存在两大缺陷：首先，Matplotlib 是一个静态的绘图模块，即我们绘出的图像是静态的，就像用看图软件打开图片一样，没有网页绘图的交互式效果；其次，Matplotlib 绘图结果的分享很不方便，在绘图结果分享给别人时只能以图片的方式分享，别人看到的绘图结果完全是静态的，分享体验很不好。Matplotlib 一直以来都是 Python 可视化的主力军，但是确实存在无法克服的缺陷，并且其他的 Python 绘图模块如 Ggplot、Bokeh、Pygal 等都比较小众，绘图功能比较单一，完成不了对 Matplotlib 的替代。

为了解决 Python 在可视化中存在的问题，Plotly 应运而生，它是一个基于 JavaScript 的动态绘图模块。Plotly 的绘图效果与我们在网页上看到的动态交互式绘图效果是一样的，其默认的绘图结果是一个 HTML 网页文件，通过浏览器就可以查看。我们可以把这个 HTML 文件分享给其他人，对方看到的效果与我们在本机上看到的效果完全一样。

Plotly 有着自己强大又丰富的绘图库，支持各种类型的绘图方案。Plotly 是基于 JavaScript 的绘图库，所以其绘图结果可以与 Web 应用无缝集成。总之，Plotly 在绘图模块上是 Matplotlib 强有力的竞争对手，Plotly 绘图的种类丰富、效果美观、易于保存与分享，因而越来越受数据分析人士的喜爱，至少笔者对 Plotly 的喜爱胜于对 Matplotlib 的喜爱。

Plotly 最初是一款商业化的绘图软件，在 2015 年 11 月 12 日，Plotly 开发团队决定把该模块的核心框架 plotly.js 开源，由此 Plotly 得到了快速发展。虽然在 2016 年 6 月，Plotly 开发团队才正式发布其 Python-api 文档，在 2017 年 1 月，Plotly 1.0 才正式发布，但是这些都阻止不了程序员对 Plotly 的喜爱。自 plotly.js 开源之后，我们可以使用本地的离线模式进行绘图，不依赖于官方的服务器，使得绘图速度更快，效果与在线绘图一样，这也是目前使用 Plotly 绘图的主流模式。

市面上有很多关于 Matplotlib 的可视化绘图教程，但是还没有 Plotly 的相关图书，作为一款非常优秀的可视化绘图模块，市面上急需一本科普性的图书。在本书创作之前，市面上就已经出现了电子版的对 Plotly 的简单翻译版本《Plotly 中文说明

1 期》，这是极宽量化开源团队在 2017 年 1 月的作品。极宽量化开源团队是一群研究“Python 量化投资”的爱好者自愿组成的一个团队，该团队成立的初衷是为国内量化投资领域做出自己的一份贡献，目前已经成功初步翻译 PyAlgoTrade、Seaborn、StatsModels、Plotly 等开源模块，并公开上传到网络上，《Plotly 中文说明 1 期》正是该团队的一个作品。

我作为极宽量化开源团队《Plotly 中文说明 1 期》项目的第一负责人，最初的想法只是单纯地把 Plotly 的基础内容简单翻译一下，以最简单、最快速的方式呈现给大家，方便大家使用。但是后来电子工业出版社的黄爱萍编辑找我沟通，请我编写一本 Plotly 数据可视化的相关图书，她认为 Plotly 发展很快，市场上需要一本 Plotly 的相关教材。经过一段时间的权衡，考虑到个人对 Plotly 的掌握程度、开源团队对 Plotly 的热情、个人在写《PyQt 5 快速开发与实战》时与黄编辑建立的良好关系，以及《Plotly 中文说明 1 期》存在的太多缺陷等，也为了能让更多的人接触 Plotly 这个优秀的绘图模块，于是决定再次抽出大量的时间来完成本书的创作，这就是本书的写作背景。

Plotly 是一个非常优秀的顶级绘图模块，如此优秀的开源模块在国内的知名度却不是很高，这对国内开发人员来说是一个很大的遗憾。顶级模块在特定的领域达到家喻户晓的程度是一个必然的趋势，Plotly 正是这种模块，它在可视化绘图领域的表现终会大放异彩。虽然目前 Plotly 在国内知名度不是很高，但其在可视化绘图领域做到家喻户晓是一个必然的趋势，只是需要有人加速这种趋势的演化过程，这就是本书存在的意义。

本书结构

本书的框架结构如下。

第 1 章是本书的快速入门部分，介绍 Plotly 的安装环境，对在线绘图与离线绘图做了简要的介绍。

第 2 章是基础绘图部分，对 Plotly 的一些常见的基础图形如线形图、柱状图、饼图、气泡图和直方图等做了一些介绍。

第 3 章是高级图形部分，对 Plotly 的时间序列绘图、表格绘图、多个坐标轴绘图、多子图绘图、SVG 绘图等做了一些介绍，是 Plotly 绘图的高级应用。

第 4 章是 Pandas 部分，介绍 Pandas 这个顶级数据分析模块使用 Plotly 进行绘图的方法。

第 5 章是金融绘图部分，主要为金融领域的特殊图形尤其是 K 线图的绘制提供了解决方案。

第 6 章是 Matplotlib 部分，主要介绍如何把 Matplotlib 绘图迁移到 Plotly 中。

第 7 章是网页开发部分，主要介绍 Plotly 在 Python 网页开发框架 Django 和 Flask 中的应用。

第 8 章是 GUI 开发部分，主要介绍 Plotly 在 GUI 开发框架 PyQt 5 中的应用。

第 9 章是机器学习部分，主要介绍 Plotly 在机器学习框架 Sklearn 与 PyTorch 中的应用。

第 10 章是量化投资部分，主要介绍 Plotly 在量化投资领域中的可视化应用。

第 11 章是其他语言应用部分，主要对其他语言如 R、MATLAB、JavaScript 的 Plotly 绘图做了简要的介绍。

如果你仅仅对 Plotly 的基础绘图有兴趣，那么前两章的内容就能满足你的需求；如果你对 Plotly 更高级的绘图有兴趣，那么可以参考第 3 章的内容；对于本书其他章节的内容，可以根据自己的实际情况有选择地阅读，毕竟 Plotly 绘图所涉及的范围特别广泛，并不是每个人对这些领域都同时感兴趣。

本书代码与交流

本书的所有代码都将保存在 GitHub 上，后续代码更新也会以 GitHub 地址为准，网址是 https://github.com/sunshe35/PythonPlotlyCodes，读者可自行下载。另外，为了方便读者交流、学习 Plotly，笔者建立了 QQ 群（群号：72203080）。

致谢

Plotly 虽然只是一个绘图模块，但是其应用场景非常广泛，除有 Matplotlib 的基本绘图功能外，其在 Web 开发、GUI 开发、机器学习、量化投资等领域也有很好的应用场景。由于其应用场景特别广泛，结合笔者自身知识的局限性，所以写好一本书需要多个人的共同努力，非常感谢下面这些作者对本书的创作所付出的努力：王硕负责本书的 Flask、PyTorch、JavaScript 部分；邢梦来负责本书的 Matplotlib 部分；

袁泉负责本书的基础绘图与 Sklearn 部分；吴娜负责本书的 Pandas 部分，其他都由本人孙洋洋负责。在 Plotly 的写作过程中，还有一些人为本书的顺利出版做出了贡献：首先，感谢《Plotly 中文说明 1 期》的开源组成员们，你们的贡献是本书基础部分的雏形，相关人员的网名和 QQ 号分别是 youngle sunny 1535327967、余勤 441499022、华子 32509167、啦啦啦 505512828、禛 948280670、L. 1248515039、Rikimaru 11766429、iris 704699640、信平 759949947、吴娜 2184934、周涛 510548099、zw 木子 719735825、非洲兔 85011284、十二月 378258849、大朱 775941748、我爱作文你信吗 571171954。其次，感谢山西证券金融科技部的陈亦苏、成都数联铭品科技有限公司的刘赣，以及极宽量化开源组的梁勇对本书网页开发部分提供的帮助，感谢西南财经大学统计学院王彦锋博士对本书 R 语言部分提供的帮助，感谢北京大学汇丰商学院硕士研究生扶禄城对本书 MATLAB 部分提供的帮助。再次，感谢黄爱萍与戴新两位编辑对本书的出版所付出的努力。最后，感谢我的父母与兄弟姐妹对我的关心与照顾，我现在取得的成果离不开你们对我的付出。

与读者相识于 Plotly 是一种缘分，能够看到本书说明读者对 Plotly 是感兴趣的，感谢读者愿意花费时间阅读本书，希望每一位读者都能够通过阅读本书有所收获，真心祝愿你们都能够学习顺利、事业有成。

孙洋洋

2018 年 4 月

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **下载资源**：本书如提供示例代码及资源文件，均可在 下载资源 处下载。
- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：*http://www.broadview.com.cn/34113*

目　　录

第 1 章　快速开始 …… 1

1.1　Plotly 简介 …… 1

1.2　安装与安装环境 …… 6

1.3　在线初始化 …… 6

1.4　在线绘图隐私说明 …… 8

1.5　开始在线绘图 …… 8

1.6　使用离线绘图库 …… 11

1.7　参数解读 …… 13

1.8　查看帮助 …… 14

第 2 章　Plotly 基础图形 …… 16

2.1　基础案例解读 …… 17

2.2　基本绘图流程 …… 19

2.3　散点图 …… 20

2.3.1　基本案例 …… 20

2.3.2　样式设置 …… 22

2.3.3　应用案例 …… 24

2.3.4　参数解读 …… 26

2.4　气泡图 …… 28

2.4.1　基本案例 …… 28

2.4.2 样式设置 ····· 29
2.4.3 缩放设置 ····· 30
2.4.4 参数解读 ····· 33
2.5 线形图 ····· 33
2.5.1 基本案例 ····· 33
2.5.2 数据缺口与连接 ····· 35
2.5.3 数据插值 ····· 38
2.5.4 填充线形图 ····· 41
2.5.5 应用案例 ····· 45
2.5.6 参数解读 ····· 49
2.6 柱状图 ····· 49
2.6.1 基本柱状图 ····· 49
2.6.2 柱状簇 ····· 50
2.6.3 层叠柱状图 ····· 52
2.6.4 瀑布式柱状图 ····· 54
2.6.5 图形样式设置 ····· 56
2.6.6 应用案例 ····· 58
2.6.7 参数解读 ····· 60
2.7 水平条形图 ····· 61
2.7.1 基本案例 ····· 61
2.7.2 应用案例 ····· 64
2.7.3 参数解读 ····· 70
2.8 甘特图 ····· 70
2.8.1 基本甘特图 ····· 70
2.8.2 甘特图（按数字索引） ····· 71
2.8.3 甘特图（按类别索引） ····· 72
2.8.4 应用案例 ····· 74
2.9 面积图 ····· 76
2.9.1 基本面积图 ····· 76
2.9.2 内部填充面积图 ····· 78
2.9.3 堆积面积图 ····· 80

2.10 直方图 83
2.10.1 基本直方图 83
2.10.2 重叠直方图 84
2.10.3 层叠直方图 85
2.10.4 累积直方图 87
2.10.5 应用案例 88
2.10.6 参数解读 89
2.11 饼图 90
2.11.1 基本饼图 90
2.11.2 环形饼图 91
2.11.3 样式设置 92
2.11.4 应用案例 93
2.11.5 参数解读 98
2.12 更多案例 99
2.13 Plotly 对象概览 100
第 3 章 Plotly 高级图形 110
3.1 时间序列 110
3.1.1 使用方法 110
3.1.2 时间范围约束 111
3.2 滑动选择控件 113
3.3 表格 117
3.3.1 入门案例 117
3.3.2 添加链接 118
3.3.3 使用 Pandas 120
3.3.4 改变大小与颜色 121
3.3.5 表格与图 124
3.4 多图表 129
3.5 多个坐标轴 130
3.5.1 双坐标轴 130
3.5.2 多坐标轴 132

3.5.3 共享坐标轴……136
3.6 多子图……138
3.6.1 双子图（方法一）……138
3.6.2 双子图（方法二）……139
3.6.3 多子图（方法一）……141
3.6.4 多子图（方法二）……142
3.6.5 分割视图区间……144
3.6.6 子图共享坐标轴（方法一）……147
3.6.7 子图共享坐标轴（方法二）……149
3.6.8 子图坐标轴自定义……152
3.6.9 嵌入式子图……154
3.6.10 混合图……155
3.7 绘制 SVG……159
3.7.1 线形图的绘制……160
3.7.2 线形图应用：创建图形的切线……163
3.7.3 矩形图的绘制……166
3.7.4 矩形图应用：设置时间序列区域高亮显示……169
3.7.5 圆形图的绘制……171
3.7.6 圆形图应用：高亮显示散点图的聚集簇……174
第 4 章 Plotly 与 Pandas……178
4.1 简单快速入门……178
4.1.1 基本线形图……179
4.1.2 基本散点图……180
4.1.3 基本柱状图……181
4.2 使用 cufflinks 绘图……183
4.2.1 安装 cufflinks……183
4.2.2 快速入门……183
4.2.3 快速获取数据……185
4.2.4 自定义绘图……186
4.2.5 常见经典图形……188

4.2.6 更多案例 ····· 193

第 5 章 金融绘图 ····· 194

5.1 快速绘制 K 线图 ····· 194

5.1.1 检查 Plotly 版本 ····· 194

5.1.2 快速绘制 OHLC（美国线）图 ····· 194

5.1.3 快速绘制蜡烛图 ····· 197

5.2 K 线图的优化 ····· 199

5.2.1 过滤非交易时间 ····· 199

5.2.2 设置形状、颜色和注释 ····· 203

5.2.3 添加技术指标 ····· 207

5.3 使用自定义数据的金融绘图 ····· 211

5.4 高级金融绘图 ····· 214

5.4.1 入门案例 ····· 214

5.4.2 综合案例 ····· 215

第 6 章 Matplotlib ····· 217

6.1 Matplotlib 简介 ····· 217

6.2 安装 Matplotlib ····· 218

6.3 调整 Matplotlib 参数 ····· 220

6.4 常用的 API 功能 ····· 222

6.5 线性函数 ····· 223

6.6 增加子图 ····· 225

6.7 确定坐标范围 ····· 228

6.8 概率图 ····· 229

6.9 散点图 ····· 232

6.10 柱状图 ····· 235

6.11 更多扩展 ····· 239

第 7 章 Plotly 与网页开发 ····· 240

7.1 Plotly 在 Django 中的应用 ····· 240

7.1.1 安装环境搭建 ····· 240

7.1.2 安装环境测试……241
7.1.3 入门案例一……243
7.1.4 入门案例二……247
7.1.5 更多案例扩展……254
7.1.6 应用案例一……256
7.1.7 应用案例二……258
7.2 Plotly 在 Flask 中的应用……267
7.2.1 安装 Flask……269
7.2.2 最小的 Web 应用……269
7.2.3 模板渲染……270
7.2.4 入门案例一……272
7.2.5 入门案例二……275
7.2.6 应用案例……283
第 8 章 Plotly 与 GUI 开发……293
8.1 PyQt 的安装……295
8.2 案例解读……295
8.3 设置提升的窗口部件……298
8.4 Plotly_PyQt 5 的使用……300
8.5 更多扩展（Plotly）……304
8.6 Plotly 与 PyQt 5.6 的结合……305
8.7 更多扩展（Matplotlib）……309
8.8 应用案例：展示产品组合信息……309
第 9 章 Plotly 与机器学习……316
9.1 Plotly 在 Sklearn 中的应用……316
9.1.1 分类问题……316
9.1.2 回归问题……319
9.1.3 聚类问题……321
9.2 PyTorch 可视化工具……326
9.2.1 Visdom 简介……326

9.2.2 安装 Visdom ····· 327
9.2.3 Visdom 与 Plotly ····· 328
9.2.4 Visdom 基本概念 ····· 328
9.2.5 Visdom 经典案例 ····· 329
9.2.6 Visdom 与 PyTorch ····· 345
第 10 章 Plotly 在量化投资中的应用 ····· 346
第 11 章 Plotly 在其他语言中的应用 ····· 355
11.1 Plotly 在 R 语言中的应用 ····· 355
11.1.1 安装 R 语言 ····· 355
11.1.2 安装 Plotly 模块 ····· 356
11.1.3 Plotly 应用分析 ····· 356
11.1.4 更多扩展 ····· 362
11.2 Plotly 在 MATLAB 中的应用 ····· 363
11.2.1 下载与安装 ····· 363
11.2.2 基础入门 ····· 365
11.2.3 经典案例 ····· 367
11.2.4 更多扩展 ····· 376
11.3 Plotly 在 JavaScript 语言中的应用 ····· 377
11.3.1 基础入门 ····· 377
11.3.2 散点图 ····· 380
11.3.3 条形图 ····· 383
11.3.4 扇形图 ····· 384
11.3.5 更多扩展 ····· 386

第 1 章

快速开始

1.1 Plotly简介

Pandas（潘达思）数据分析软件和 Plotly 互动式绘图模块可以称为 Python 数据分析和 Python 量化分析两大重量级模块库。

事实上，这两者的影响力早已超越 Python 语言领域。

- Pandas 目前已经被业界公认是大数据工程一线最好的解决方案。
- Plotly 被称为史上最好的绘图工具之一，是 GitHub 开源网站最“火”的两大新项目之一（另外一个是 Netflix 的 Spinnaker 云计算项目）。

为了更好地展示金融数据的复杂性，下面首先介绍 Plotly 绘图模块。如图 1-1 所示是 Plotly 网站截图，网址是 https://plot.ly。

Python 量化的关键是金融数据可视化，无论是传统的 K 线图，还是现在的互动策略分析，都需要大量的可视化图表。具体到编程代码，就是各种 Python 绘图模块库，传统 Python 绘图模块库有 Matplotlib、Ggplot、Seaborn、Bokeh 等。

Plotly 绘图底层使用的是 plotly.js，plotly.js 基于 D3.js、stack.gl（WebGL 组件库，由 Plotly 团队的 Mikola Lysenko 带领开发）和 SVG，用 JavaScript 在网页上实现类似 MATLAB 和 Python Matplotlib 的图形展示功能，支持数十种图形，包括 2D 和 3D 图形，交互流畅，可以满足一般科学计算的需要。该项目成功后，开始向其他语言

移植，目前已经有 Python、MATLAB、R 语言、Jupyter 等多种版本的 API 接口。

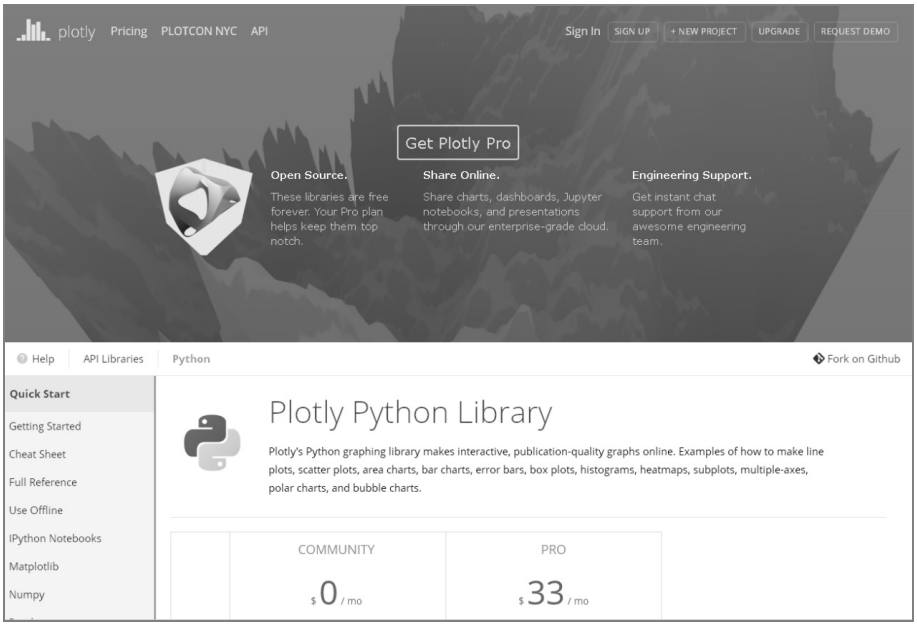

图 1-1　Plotly 网站截图

Plotly 项目的创始人是 Alex Johnson，是哈佛大学物理博士，曾经做过燃料电池。该项目的参与者中有一位华裔女生，叫 Baobao Zhang（张宝宝），是耶鲁大学政治学博士。Fernando Perez（IPython Notebook 创始人）是该项目的顾问。

Plotly 项目团队在 2013 年就已经拿到数百万美元的投资，并在加拿大蒙特利尔注册了公司，公司名称是科学数据云服务公司，号称是“首个科学数据平台”，提供线上和联机服务，方便用户使用分享、评论等功能，其投资商有 MHS 资本、西门子风险资本、Rho Ventures、Real Ventures 和硅谷银行等。

Plotly 公司有很多大牌客户，如西门子、Google、美国国家航天局（NASA）、美国空军、华盛顿邮报、美国国家卫生研究院等。

Plotly 公司的联合创始人 Jack Parmer 这样说过：对于开发者来说，绘制出漂亮的、交互式的图表并将之分享是一件很容事的事，但是对于一般的用户则很困难。Plotly 改变了这种局面，借助互联网的媒介、利用世界级的绘图分析工具，每个人都可以参与其中。

Hacker News（极客新闻）网站曾经专门讨论过各种图形模块库，如 Matplotlib、

MPLD3、Bokeh、Highcharts、Flotcharts、MetricsGraphics、Vega、Chartjs、FusionCharts 和 JSXGraph。相比而言，Plotly 功能齐全，性能显著超过其他同类产品。

Plotly 原本是收费的商业软件，幸运的是，2016 年 6 月，Plotly 绘图模块库提供了免费的社区版本，并新增了 Python 等多种编程语言的接口，以及离线模式支持，这对于广大中国用户而言，获得了现实的技术支持。

Python 绘图模块库数不胜数，其中经典的有数十个。目前，Plotly 作为新一代互动型量化绘图库，基本上可以说一统天下。如图 1-2 所示是 Plotly 绘图模块库部分内置的图形类型。

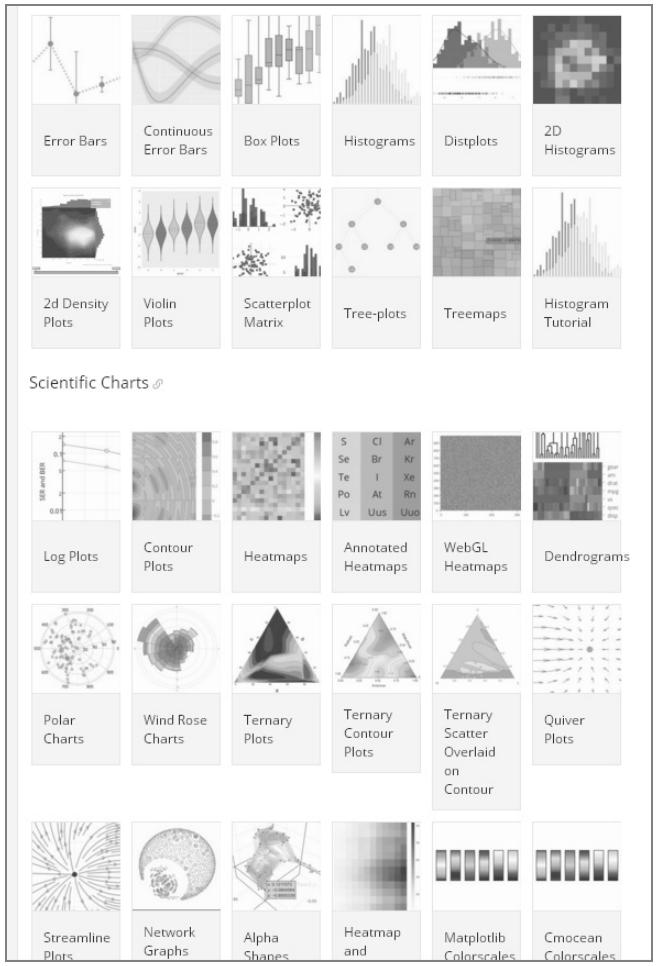

图 1-2　Plotly 绘图模块库部分内置的图形类型

笔者在编写本书时，特别针对多种 Python 绘图模块库进行了测试，测试结果表明，Plotly 绘图模块库的确是新一代 Python 绘图模块的王者之选，也是各种 Web 平台的优先选择绘图模块。

以笔者个人的测试感觉而言，Plotly 绘图模块库既有 Matplotlib 绘图模块库的强大与灵活，又有 Seaborn 统计绘图模块库的现代配色组合与优雅报表形态。

Plotly 绘图模块库可直接生成 PNG 等图像文件，与 Bokeh 绘图模块库和各种基于 Web 的 JavaScript 图表模块库类似，生成的是一个内置 JavaScript 脚本的 HTML 网页文件，虽然文件比 Bokeh 绘图模块库生成的文件略大，但在互动性方面，Plotly 绘图模块库强大得多。

在 3D 图表方面，虽然笔者没有实际测试，但基于 Plotly 网站的案例可以看出，相对传统 Python 3D 图表模块库，无论在图表种类格式，还是色彩组合方面，Plotly 绘图模块库都显得更加灵活和强大。

图 1-3 所示是 Plotly 绘图模块库绘制的 K 线图，支持互动功能，读者可以参考 Top 极宽量化社区的演示网页（http://www.ziwang.com/zwdemo.htm）。

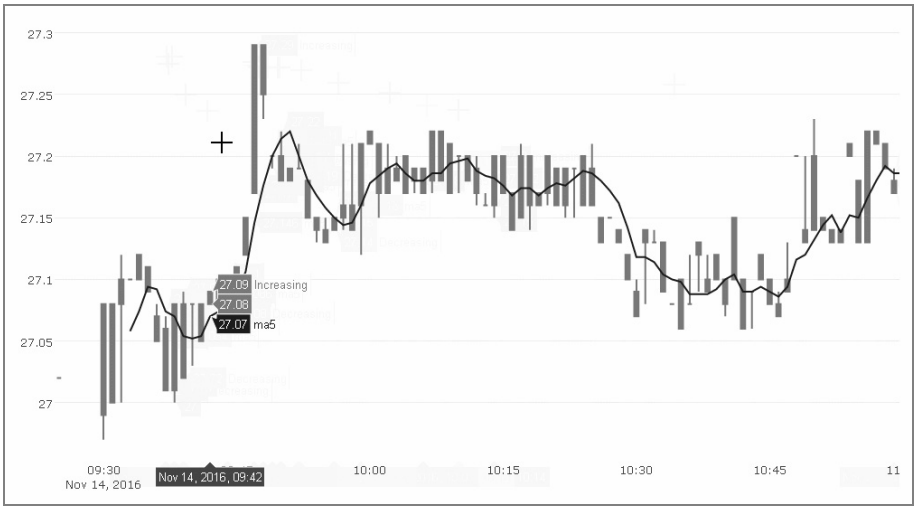

图 1-3　Plotly 绘图模块库绘制的 K 线图

请注意，案例中没有使用任何图片，是纯 JavaScript 脚本函数，所有的图形、数据和互动都是通过 JavaScript 函数完成的，这些 JavaScript 函数全部是通过 Python 自动生成的。

图 1-3 虽然是静止的，但在浏览器中，其实是纯 JavaScript 脚本的函数图形，支持各种互动功能，读者可以好好体会一下，并且看看网页源码。

Plotly 原本是基于 JavaScript 的数据图表分析绘制模块库，在编程的灵活性和图表的丰富性方面非常强大，优点数不胜数。

- Plotly 本身是一款独立的 Web 版数据可视化工具，界面友好，提供强大的互动性操作。
- 基于现代的配色组合和图表形式，相比 Matplotlib、R 语言的图表，更加现代和绚丽。
- 具有简单且强大的 3D 图表绘制功能，支持多种格式。
- 对图形参数的修改十分简单、直观，便于初学者使用。
- 有 Python、R、MATLAB、Jupyter、Excel 等多种版本的接口。
- 与 Pandas（潘达思）数据分析软件无缝集成，并提供了专门的 Plotly 绘图模块库，设计的图表非常吸引人，而且具有高度互动性，这得益于其完善的文档和简单的 Python API，用户入门也很容易，

目前，Plotly 绘图模块库支持的图表格式如下。

- 基本图表：20 种。
- 统计和海运方式图：12 种。
- 科学图表：21 种。
- 财务图表：2 种。
- 地图：8 种。
- 3D 图表：19 种。
- 报告生成：4 种。
- 连接数据库：7 种。
- 拟合工具：3 种。
- 流动图表：4 种。

随着 Plotly 绘图模块库软件的升级与更新，未来会增加更多的图表格式。

Plotly 底层使用的是 plotly.js，支持 Python、R、MATLAB、JavaScript 这四种语言的扩展，由于 Python 正处于数据分析领域的领先地位，并且其重要性呈日渐上升之势，所以本书介绍 Plotly 时以 Python 语言（Python 3）为主，同时也会给出 Plotly 在其他语言中的一些应用。

Plotly 发展很快，其应用的领域也非常广泛，不但应用在传统意义上的数据分

析与处理上，而且在网页开发、程序 GUI、机器学习等多个方面也有重要的应用。由于这个原因，本书涉及的内容很广，除 Plotly 基础知识外不会做太多深入的介绍，仅进行入门介绍，读者可以结合基础知识部分进行更深入的研究，这对于绝大部分读者来说并不是什么难事。

对于本书前半部分的基础知识内容，并不需要读者对 Python 有太多的了解，没有学过 Python 的读者只需要花费两个小时了解 Python 的基础语法就可以了。本书后半部分的内容，读者可以按照自己的实际需求阅读相关的章节，然后稍微修改一下就可以进行实战，通过这种方式快速掌握这些技能。

1.2 安装与安装环境

安装 Plotly 的方法特别简单，直接用 pip 命令安装即可，代码如下。

```
pip install plotly
```

Plotly 的 Python 包经常被更新，若需要升级到最新版，运行如下代码。

```
pip install plotly --upgrade
```

本书涉及的模块都是一些比较流行的模块，一般情况下都可以通过 pip 命令进行安装。这里笔者建议使用 Python 3.5 的 anaconda 或 winpython 开发环境，因为目前笔者使用 Python 3.6 经常遇到一些 bug，而使用 Python 3.5 则没有这些问题。下载 anaconda 或 winpython 后，默认环境就是本书所依赖的环境，如果有些模块没有找到，可以通过 pip 命令安装。

1.3 在线初始化

Plotly 提供了一个在线托管绘图结果的 Web 服务平台，用户可以在网站（https://plot.ly）免费创建一个账号。用户所绘制的图表会保存在自己的在线 Plotly 账户中，并且可以控制相应的权限。免费的托管方式是公共的，也就是自己的绘图结果其他人也能够看见。网站也提供私有的托管方式，不过是收费的，这里不推荐，有兴趣的读者可以去官网查看。

安装好 Python 和 Plotly 后，要使用 Plotly，首先要设置自己的凭据，这个凭据

在网站上注册一个账号就可以获取。当用户注册 Plotly 账号并进行登录后，在右上角可以找到自己的用户选项，单击其中的 Settings 选项，然后找到 API Key 选项，如图 1-4 所示。

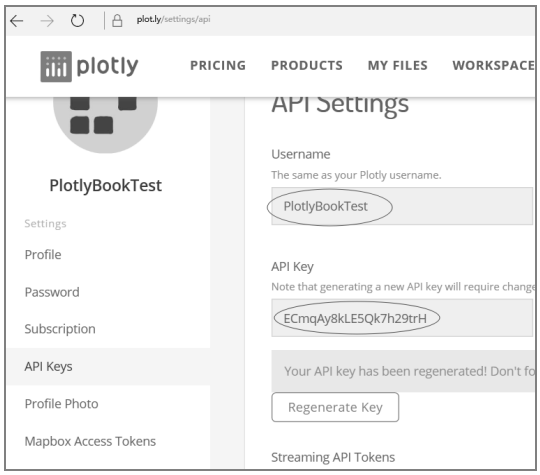

图 1-4　获取凭据

图 1-4 中的 Username 和 API Key 是笔者为编写本书注册的一个测试账号，读者可以通过注册自己的账号得到类似的结果。

下面我们把上面的 Username 和 API Key 记录下来，放入如下代码中，就完成了自己的凭据设置，由此，在线初始化完成。

```
    import plotly
    plotly.tools.set_credentials_file(username='PlotlyBookTest', api_key=
'ECmqAy8kLE5Qk7h29trH')
```

上面的初始化步骤会在当前用户目录中存放一个特殊文件.plotly/.credentials，这个文件内容看起来是下面这样的：

```
    {
        "proxy_username": "",
        "proxy_password": "",
        "api_key": "ECmqAy8kLE5Qk7h29trH",
        "stream_ids": [],
        "username": "PlotlyBookTest"
    }
```

1.4 在线绘图隐私说明

使用在线绘图，有三种不同的隐私设置类型：公有、私有和秘密。

- 公有

（1）任何人都可以查看用户的图表。

（2）显示在用户的个人资料中，可以被搜索引擎搜索到。

（3）用户不需要登录 Plotly 就可以查看这些图表。

- 私有

（1）只有自己才可以查看此图表。

（2）不会显示在 Plotly 的信息源、用户的个人资料和搜索引擎中。

（3）用户需要登录才能查看这些图表。

- 秘密

（1）任何拥有此秘密链接的人都可以查看此图表。

（2）不会显示在 Plotly 的信息源、用户的个人资料和搜索引擎中。

（3）如果它嵌入 Web 或 Jupyter Notebook 中，任何查看该网页的人都能够看到此图表。

（4）用户不需要登录就可以查看这些图表。

在默认情况下，所有图表都被设置为公有模式。免费账户（公有模式的账户）可以有一个私有图表。如果用户有额外的私有存储需求，就需要去官网升级产品服务，感兴趣的读者可以去官网查看。

1.5 开始在线绘图

在线绘图时，绘图和数据都保存在自己的云账户中。有两个方法可以实现在线绘图：py.plot()和 py.iplot()。这两个方法的作用都是在自己的账户中新建一个网址并存储绘图结果。当然，这两个方法也有不同的地方。

- 使用 py.plot()方法进行绘图会返回一个网址，可以选择是否打开这个网址，默认是打开的。
- 使用 py.iplot()方法进行绘图会返回 PlotlyDisplay 对象，并以嵌入的形式在 Jupyter Notebook 中显示出来。

下面开始讲解我们的第一个案例（见文件 Chapter01/first_start.py），只需运行如下代码。

```
import plotly.plotly as py
from plotly.graph_objs import *
trace0 = Scatter(
    x=[1, 2, 3, 4],
    y=[10, 15, 13, 17]
)
trace1 = Scatter(
    x=[1, 2, 3, 4],
    y=[16, 5, 11, 9]
)
data = Data([trace0, trace1])
py.plot(data, filename = 'first_start')
```

运行上面的代码会打开一个网页，需要稍等片刻才能渲染完成，我们会看到如图 1-5 所示的结果。

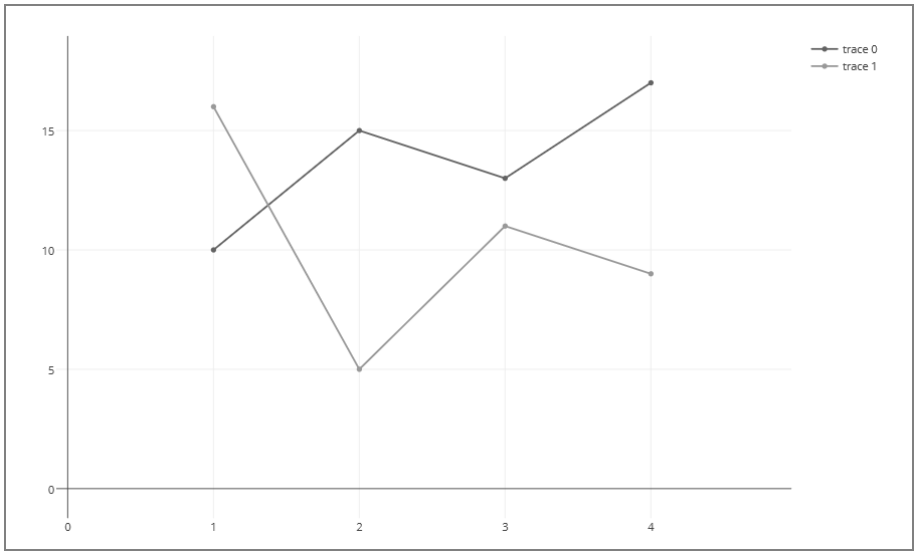

图 1-5　案例运行结果

本书的第一个案例代码运行完成。

同样，在 Jupyter Notebook（如果读者不知道什么是 Jupyter Notebook，可以跳

过这部分）中，我们也可以在 Spyder 等 IDE 里面完成相同的绘图操作，只需要把 py.plot()改成 py.iplot()就可以了（见文件 Chapter01/first_start.ipynb），代码如下。

```
import plotly.plotly as py
from plotly.graph_objs import *

trace0 = Scatter(
    x=[1, 2, 3, 4],
    y=[10, 15, 13, 17]
)
trace1 = Scatter(
    x=[1, 2, 3, 4],
    y=[16, 5, 11, 9]
)
data = Data([trace0, trace1])
py.iplot(data, filename = 'basic-line')
```

运行上面的代码后，结果如图 1-6 所示。

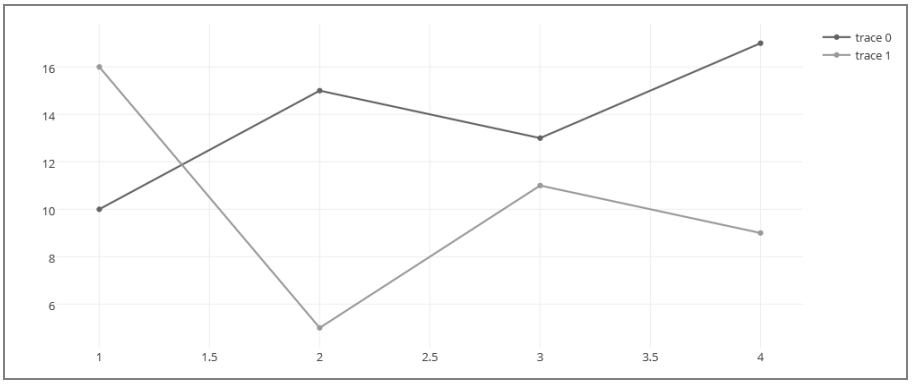

图 1-6　案例运行结果

我们看到，这个案例在 Jupyter Notebook 中绘图成功。

注意

有些读者可能是第一次接触 Jupyter Notebook，不知道如何导入.ipynb 文件，这里简单介绍一下。当我们打开 Jupyter Notebook 后，默认情况下，在当前目录中找不到案例文件 first_start.ipynb，可以按以下步骤导入这个文件。

打开 Jupyter Notebook 网页后，单击“Upload”按钮，如图 1-7 所示。这时会弹出一个文件管理器窗口，选择文件 first_start.ipynb，然后再次单击“Upload”按钮，就会在当前目录下看到 first_start.ipynb 文件，双击该文件打开即可。

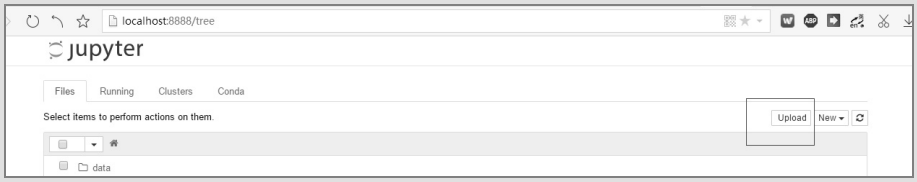

图 1-7　加载文件

1.6　使用离线绘图库

通过前面的案例，我们可以看到 Plotly 的绘图结果虽然美观，但是绘图速度太慢，原因是 Plotly 的服务器在国外，国内用户使用 Plotly 的在线绘图会有网络延迟。如果仅有在线绘图，恐怕 Plotly 不会发展得这么快，一是如果用户都使用在线绘图，会导致官方的服务器压力增大，这样免费服务无法长久维持下去或体验越来越差；二是这种绘图速度国内用户是无法接受的，所以开源的动态绘图模块 Plotly 支持离线绘图功能。

Plotly 的离线绘图功能允许在没有网络的情况下绘图，并把图像保存到本地。有两种方法可以实现上述功能：plotly.offline.plot ()和 plotly.offline.iplot ()。

- 使用 plotly.offline.plot ()方法会在本地新建一个 HTML 文件，并可以选择是否在浏览器中打开这个文件。
- 使用 plotly.offline.iplot ()方法会在 Jupyter Notebook 中直接绘图，而不需要新建一个 HTML 文件。

案例代码如下（见文件 Chapter01/first_offline_start.py）。

```
import plotly as py
from plotly.graph_objs import Scatter, Layout, Data

trace0 = Scatter(
    x=[1, 2, 3, 4],
    y=[10, 15, 13, 17]
```

```
)
trace1 = Scatter(
    x=[1, 2, 3, 4],
    y=[16, 5, 11, 9]
)
data = Data([trace0, trace1])

py.offline.plot(data, filename = 'first_offline_start')
```

运行以上代码，结果如图 1-8 所示。

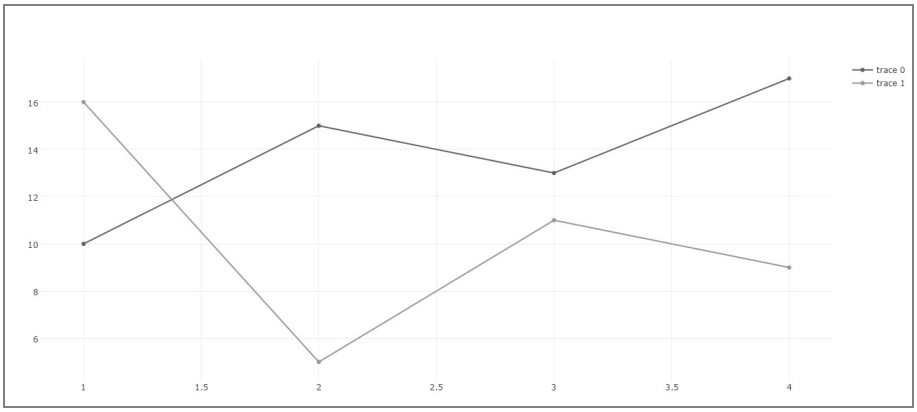

图 1-8　案例运行结果

在 Jupyter Notebook 中需要注意的是，这里需要添加一个初始化步骤，在开始绘图之前要加入一行代码。

```
plotly.offline.init_notebook_mode()
```

详细代码如下（见文件 Chapter01/first_offline_start.ipynb）。

```
import plotly as py
from plotly.graph_objs import Scatter, Layout, Data

py.offline.init_notebook_mode()

trace0 = Scatter(
    x=[1, 2, 3, 4],
    y=[10, 15, 13, 17]
```

```
)
trace1 = Scatter(
    x=[1, 2, 3, 4],
    y=[16, 5, 11, 9]
)
data = Data([trace0, trace1])

py.offline.iplot(data, filename = 'first_offline_start')
```

代码运行结果如图 1-9 所示，与上一个案例的输出效果一样。

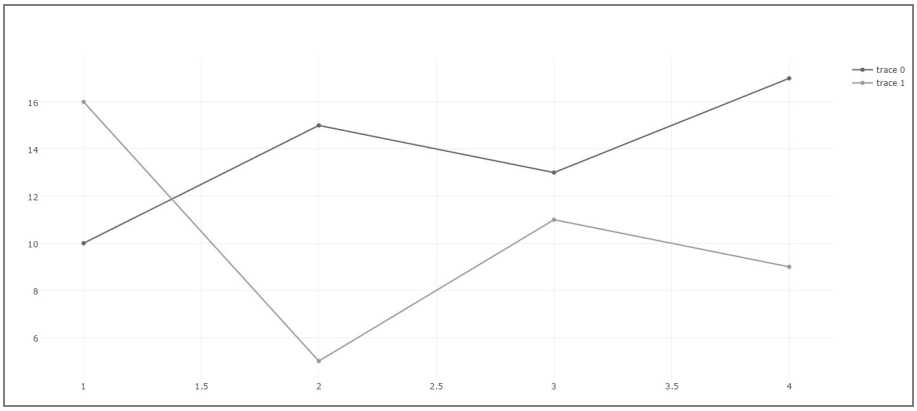

图 1-9　案例运行结果

由于在线绘图和离线绘图的绘图结果没有什么不同，所以对于在线绘图，读者可以自己动手修改代码。

1.7　参数解读

py.plot 是绘制图形的主函数，其主要参数如下。

- show_link：默认为 True，显示右下角的链接。
- link_text：右下角显示的文字，默认为 Export to plotly.ly。
- validate：默认为 True，确保所有关键字是有效的。但是当需要额外、非必需的关键字或 plotly.js 版本比 graph_reference.json 版本旧时，会忽略这部分内容。
- filename：设置绘图结果的存储路径，默认为 temp-plot.html。

1.8 查看帮助

查看帮助的操作很简单，只需要如下简单的两行代码。

```
# 使用在线绘图查看帮助的方法
import plotly.plotly as py
help(py.plot)
```

对于使用 Jupyter Notebook 的读者，可以使用如下代码查看帮助。

```
# 使用 Jupyter Notebook 在线绘图查看帮助的方法
import plotly.plotly as py
help(py.iplot)
```

对于离线绘图模式，有如下两种查看帮助的方法。

```
# 使用离线绘图查看帮助的方法
import plotly
help(plotly.offline.plot)
# 使用 Jupyter Notebook 离线绘图查看帮助的方法
import plotly
help(plotly.offline.iplot)
```

其中一个查看帮助的结果代码如下。

```
    Signature: plotly.offline.iplot(figure_or_data, show_link=True,
link_text='Export to plot.ly', validate=True, image=None, filename=
'plot_image', image_width=800, image_height=600, config=None)Docstring:
    Draw plotly graphs inside an IPython or Jupyter notebook without
    connecting to an external server.
    To save the chart to Plotly Cloud or Plotly Enterprise, use
    `plotly.plotly.iplot`.
    To embed an image of the chart, use `plotly.image.ishow`.

    figure_or_data -- a plotly.graph_objs.Figure or plotly.graph_objs.Data
or
                      dict or list that describes a Plotly graph.
                      See https://plot.ly/python/ for examples of
                      graph descriptions.
```

```
    Keyword arguments:
    show_link (default=True) -- display a link in the bottom-right corner of
                            of the chart that will export the chart to
                            Plotly Cloud or Plotly Enterprise
    link_text (default='Export to plot.ly') -- the text of export link
    validate (default=True) -- validate that all of the keys in the figure
                           are valid? omit if your version of plotly.js
                           has become outdated with your version of
                           graph_reference.json or if you need to include
                           extra, unnecessary keys in your figure.
    image (default=None |'png' |'jpeg' |'svg' |'webp') -- This parameter sets
       the format of the image to be downloaded, if we choose to download
an
       image. This parameter has a default value of None indicating that no
       image should be downloaded. Please note: for higher resolution images
       and more export options, consider making requests to our image servers.
       Type: `help(py.image)` for more details.
    filename (default='plot') -- Sets the name of the file your image
       will be saved to. The extension should not be included.
    image_height (default=600) -- Specifies the height of the image in `px`.
    image_width (default=800) -- Specifies the width of the image in `px`.
    config (default=None) -- Plot view options dictionary. Keyword arguments
       `show_link` and `link_text` set the associated options in this
       dictionary if it doesn't contain them already.

    Example:
    ```
    from plotly.offline import init_notebook_mode, iplot
    init_notebook_mode()
    iplot([{'x': [1, 2, 3], 'y': [5, 2, 7]}])
    # We can also download an image of the plot by setting the image to the
    format you want. e.g. `image='png'`
    iplot([{'x': [1, 2, 3], 'y': [5, 2, 7]}], image='png')
    ```File:     d:\anaconda3\lib\site-packages\plotly\offline\offline.
pyType:     function
```

第 2 章

Plotly 基础图形

Plotly 绘图模块库支持的图形格式很多，其绘图对象包括如下几种。

- Angularaxis：极坐标图表。
- Area：区域图。
- Bar：条形图。
- Box：盒形图，又称箱线图、盒子图、箱图。
- Candlestick 与 OHLC：金融行业常用的 K 线图与 OHLC 曲线图。
- ColorBar：彩条图。
- Contour：轮廓图（等高线图）。
- Choropleth：等值线图。
- Line：曲线图。
- Heatmap：热点图。
- Histogram：直方图。
- Histogram 2d：2D 平面直方图。
- Histogram 2d Contour：二维轮廓直方图。
- Pie：扇形图。
- Scatter：坐标分布图（包括线形图、散点图）。

本章将介绍最常用的几种图形格式：线形图、散点图、条形图、饼图、直方图、面积图、甘特图，以及金融行业常用的 K 线图等。

2.1　基础案例解读

下面通过案例介绍在 Python 语言中使用 Plotly 绘图模块库。由于使用在线绘图模式绘图速度较慢，而且使用在线绘图模式需要注册账号，所以这里只介绍离线绘图模式。

案例 2-1 的文件名是 first_start_introduction.py，讲解使用 Plotly 绘图模块库进行最简单的绘图。

为了便于逐行讲解案例，此处采用直接截图的方式展示代码，如图 2-1 所示。

```
1   # -*- coding: utf-8 -*-
2   import pandas as pd
3   import plotly as py
4   import plotly.graph_objs as pygo
5
6   trace0 = pygo.Scatter(
7       x=[1, 2, 3, 4],
8       y=[10, 15, 13, 17]
9   )
10  trace1 = pygo.Scatter(
11      x=[1, 2, 3, 4],
12      y=[16, 5, 11, 9]
13  )
14  data = pygo.Data([trace0, trace1])
15  #
16  py.offline.plot(data, filename = 'tmp/first_start_introduction.html')
17  # py.offline.plot(data)  #temp-plot.html
18  # py.offline.plot(data, filename = 'tmp/first_start_introduction.html',image='png')
19  #
20  print('ok')
```

图 2-1　案例 2-1 代码截图

案例 2-1 代码逐行说明如下。

- 第 1 行，说明该代码文件使用的是 UTF-8 编码格式，也是 Python 的例行文件头。
- 第 2～4 行，导入（import）所需的模块库。
- 第 6 行，定义一个名为 trace0 的变量，用于保存绘图数据；Plotly 一般称一个绘图对象为 trace（画痕、画轨、画迹）；每个绘图对象都由 Plotly 模块库里面的 graph_objs（图像对象）子模块的 Scatter（数据布局）对象定义；Scatter 对象的输入数据与 Plotly 绘图模块库中的许多函数和对象一样，都是字典格式，而且接受多种复杂的复合字典格式。

- 第 7～8 行，定义变量 trace0 的 *x* 和 *y* 坐标，坐标采用列表数据格式，坐标 *x*、*y* 的数据长度必须一样，不然会出现空缺坐标；其中的 *x*、*y* 字符是 Plotly 模块库 Scatter 对象的内部变量名称，类似关键字。
- 第 10～12 行，定义另外一条曲线 trace1 的坐标数据，细节参见第 6～8 行的说明。
- 第 14 行，定义一个变量名 data，通过 pygo 模块库的 Data 函数，把代表两条曲线的变量 trace0 和 trace1 定义为一组图形数据，注意 Data 函数的输入参数是列表数据格式，所以要在变量 trace0 和 trace1 外面加上表示列表的“[]”符号。
- 第 16 行，根据 data 变量的数据绘制图形。若代码无误，正确运行后，程序会自动调用浏览器，打开生成的图形，并提供互动显示，运行结果如图 2-2 所示。
- 第 20 行，运行结束后，输出“ok”信息。本程序运行后，输出的图形是通过浏览器显示的，有时浏览器在后台，不知道程序运行已经完成，所以在程序最后增加了一行运行结束的提示信息。

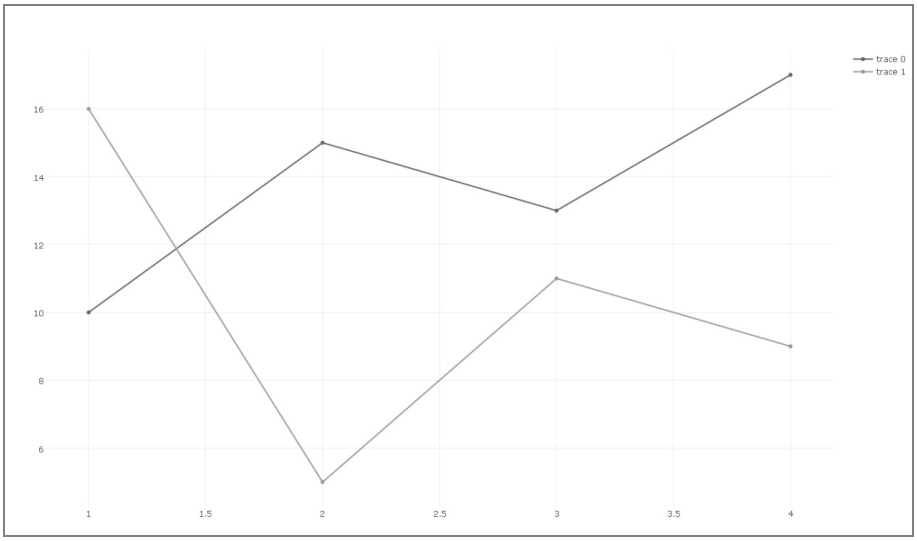

图 2-2　案例 2-1 运行结果

案例 2-1 的第 17 行和第 18 行代码表示不同的输出模式，下面详细说明。

第 16 行中的 filename 参数，文件名格式必须以“.html”为后缀，以“.htm”为后缀会出错，读者可以尝试一下。

运行第 17 行和第 18 行代码时，请先屏蔽第 16 行代码，以免混乱。

第 17 行不使用 filename 参数定义文件名，Plotly 模块库会自动在当前代码目录

下生成一个名为 temp-plot.html 的文件，并自动用浏览器打开。对于多个图形，Plotly 模块库会生成多个 HTML 文件，再用浏览器自动打开，用户需要自己设置文件名，否则新的图形文件会覆盖旧的图形文件。

第 18 行使用 image 参数，设置输出的静态图片格式为 PNG 格式，image 参数支持 PNG、JPEG、SVG、WebP 格式，默认是 None（无）。

与其他绘图模块库不同，Plotly 生成的不是图形文件，而是一个内置 JavaScript 脚本的 HTML 网页文件，最终的图像文件需要调用浏览器的渲染引擎才能生成。

用户设置 image 图形输出格式后，浏览器会弹出一个下载图片对话框，需要用户手动保存输出的图形，如图 2-3 所示。

图 2-3　下载图片对话框

可能因为 Plotly 绘图模块库重点是云端及商业用户，所以对离线用户有些限制，这也算是目前 Plotly 的一个小 bug，希望未来的版本能够改进。

2.2　基本绘图流程

经过以上分析，使用 Plotly 绘图模块库进行绘图的完整流程应该包括以下命令。

- 添加图轨数据（add_trace），使用的是 Scatter 等函数命令。

- 设置画面布局，使用 layout 命令。
- 集成图形、布局数据，命令有 Data、Figure。
- 绘制图形的输出，命令是 offline.plot，自定义的短命令是 pyplt。

这个绘图流程对绝大部分 Plotly 绘图案例都适用，接下来讲解 Plotly 的一些常见图形的绘制方法。

2.3 散点图

2.3.1 基本案例

线形图又称为曲线图，是最常用的图形类型。与传统的绘图软件不同，Plotly 没有独立的线形图函数，而是把线形图与散点图全部用 Scatter 函数实现。

下面的代码（见文件 scatter_basic_demo.py）是基本散点图的绘制方法，包括线形图与散点图两种图形的混合。

```
import plotly as py
import plotly.graph_objs as go

# ----------pre def
pyplt = py.offline.plot

# ----------code
# Create random data with numpy
import numpy as np

N = 100
random_x = np.linspace(0, 1, N)
random_y0 = np.random.randn(N)+5
random_y1 = np.random.randn(N)
random_y2 = np.random.randn(N)-5

# Create traces
trace0 = go.Scatter(
```

```
    x = random_x,
    y = random_y0,
    mode = 'markers', # 纯散点的绘图
    name = 'markers' # 曲线名称
)
trace1 = go.Scatter(
    x = random_x,
    y = random_y1,
    mode = 'lines+markers', # “散点+线”的绘图
    name = 'lines+markers'
)
trace2 = go.Scatter(
    x = random_x,
    y = random_y2,
    mode = 'lines', # 线的绘图
    name = 'lines'
)

data = [trace0, trace1, trace2]
pyplt(data, filename='tmp\scatter_basic_demo.html')
```

代码运行结果如图 2-4 所示。

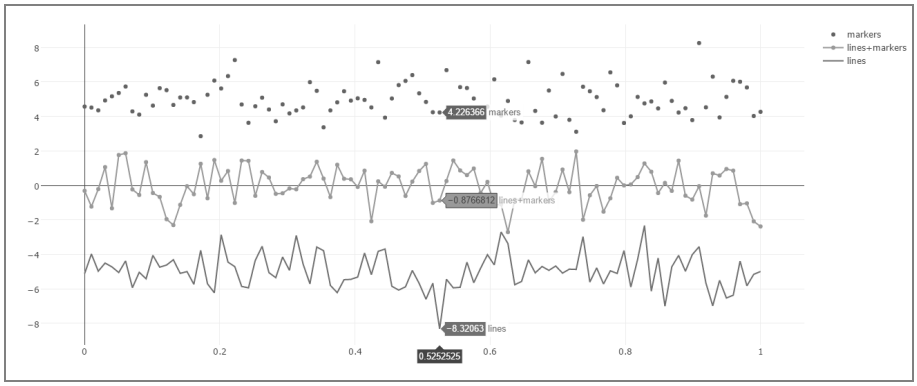

图 2-4　代码运行结果

查看案例源码可以看出，markers、lines 和 lines+markers 三个图形的输出格式不同，是因为 Scatter 函数中的 mode 参数不同：

```
mode = 'markers',#xtr1,散点图
mode = 'lines',#xtr2,线形图,
mode = 'lines + markers',#xtr3,线形图+散点图组合
```

2.3.2 样式设置

下面的代码（见文件 scatter_style.py）用来设置点的大小和颜色，以及线条的大小和颜色。

```
import plotly as py
import plotly.graph_objs as go

import numpy as np

# ----------pre def
pyplt = py.offline.plot

# ----------code
N = 500
x = np.random.randn(N)

trace0 = go.Scatter(
    x = np.random.randn(N),
    y = np.random.randn(N)+2,
    name = 'Above',
    mode = 'markers+lines',
    marker = dict(
        size = 10, # 设置点的宽度
        color = 'rgba(152, 0, 0, .8)', # 设置点的颜色
        line = dict(
            width = 2, # 设置线条的宽度
            color = 'rgb(0, 0, 0)' # 设置线条的颜色
        )
    )
)
```

```
trace1 = go.Scatter(
    x = np.random.randn(N),
    y = np.random.randn(N) - 2,
    name = 'Below',
    mode = 'markers',
    marker = dict(
        size = 10,
        color = 'rgba(255, 182, 193, .9)',
        line = dict(
            width = 2,
        )
    )
)

data = [trace0, trace1]

layout = dict(title = 'Styled Scatter',
              yaxis = dict(zeroline = True), # 显示 y 轴的 0 刻度线
              xaxis = dict(zeroline = False) # 不显示 x 轴的 0 刻度线
             )

fig = dict(data=data, layout=layout)
pyplt(fig, filename='tmp\scatter_style.html')
```

代码运行结果如图 2-5 所示。

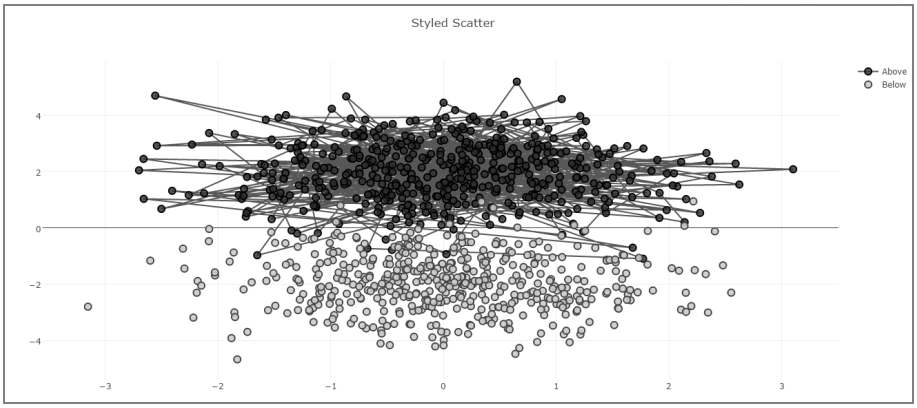

图 2-5　代码运行结果

在这个案例中，重要的是 marker 参数的设置：

```
marker = dict(
       size = 10, # 设置点的宽度
       color = 'rgba(152, 0, 0, .8)', # 设置点的颜色
       line = dict(
           width = 2, # 设置线条的宽度
           color = 'rgb(0, 0, 0)' # 设置线条的颜色
       )
   )
```

这里 size 设置的是点的宽度，width 设置的是线条的宽度，第一个 color 设置的是点的颜色，第二个 color 设置的是线条的颜色，读者可以对这些参数进行修改，实际验证一下。

需要注意的是，Plotly 中的这些参数在其他绘图函数中可以重复使用，这也是 Plotly 人性化的地方。

2.3.3 应用案例

本案例的文件名是 scatter_apply.py，讲解 Scatter 函数的使用方法，绘制曲线图与散点图，以及曲线图与散点图的组合图，代码如下。

```
# -*- coding: utf-8 -*-

import pandas as pd
import pandas as pd
import plotly as py
import plotly.graph_objs as pygo

# ----------pre def
pd.set_option('display.width', 450)
pyplt = py.offline.plot
# ----------code
df = pd.read_csv('dat/tk01_m15.csv')
df9 = df[:10];
print(df9)
```

```
    #
    idx = df9['xtim']
    xd0 = (df9['close'] - 27) * 50
    df2 = df9
    df2['xd1'] = xd0 - 10
    df2['xd2'] = xd0
    df2['xd3'] = xd0 + 10

    print('xd2\n', df2);
    # --------
    xtr1 = pygo.Scatter(
        x=idx,
        y=df2['xd1'],
        mode='markers',  # xtr1, 散点图
        name='xtr1-markers',
    )
    xtr2 = pygo.Scatter(
        x=idx,
        y=df2['xd2'],
        mode='lines',  # xtr2, 曲线图
        name='xtr2-lines',
    )
    xtr3 = pygo.Scatter(
        x=idx,
        y=df2['xd3'],
        mode='markers+lines',  # xtr3, 曲线图+散点图
        name='xtr3-markers+lines',
    )
    xdat = pygo.Data([xtr1, xtr2, xtr3])
    layout = pygo.Layout(
        title='收盘价--15 分钟分时数据',
        xaxis=pygo.XAxis(tickangle=-15),

    )
    fig = pygo.Figure(data=xdat, layout=layout,filename=r'tmp\
scatter_apply.html')
    pyplt(fig)
```

```
#
print('ok')
```

代码运行结果如图 2-6 所示。

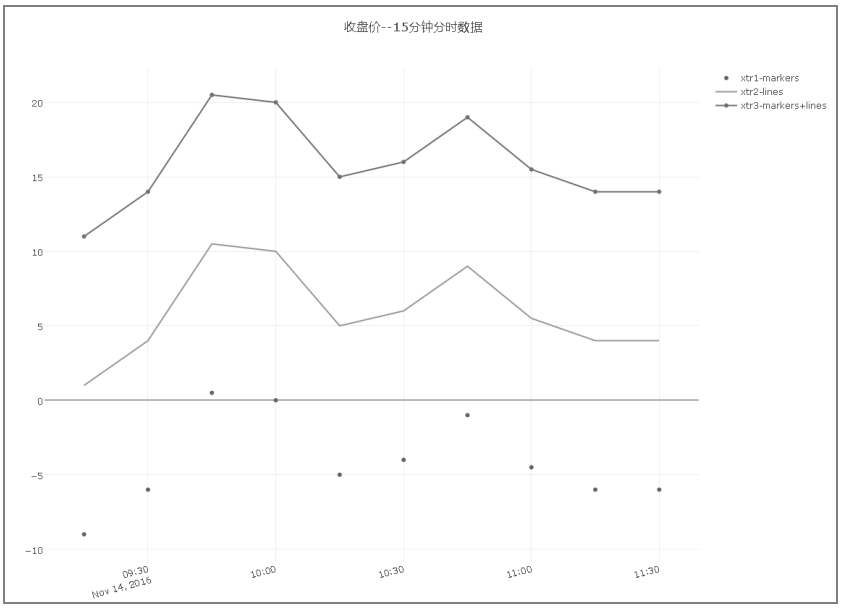

图 2-6　代码运行结果

2.3.4　参数解读

使用 Scatter 函数可以绘制线形图与散点图，主要参数如下。

- connectgaps：布尔变量，用于连接缺失数据。
- dx、dy：*x*、*y* 坐标轴的步进值，默认值是 1。
- error_x、error_y：*x*、*y* 出错信息。
- fillcolor：填充颜色。
- fill：填充模式，包括格式、颜色等。
- hoverinfo：当用户与图形互动时，鼠标指针显示的参数，包括 *x*、*y*、*z* 坐标数据，以及 text（文字信息）、name（图形名称）等参数的组合，可使用+、all、none 和 skip（忽略）作为组合连接符号，默认是 all（全部消失）。

- hoveron：当用户与图形互动时，鼠标指针显示的模式，包括 points（点图）、fills（填充图）和 points+fills（点图+填充图）三种模式。
- ids：在动画图表中，数据点和图形 key 键的列表参数。
- legendgroup：图例参数，默认是空字符串。
- line：线条参数，包括线条宽度、颜色、格式。
- marker：数据节点参数，包括大小、颜色、格式等。
- mode：图形格式，包括 lines（线形图）、markers（散点图）和 text（文本），使用+或 none 等符号进行模式组合。
- name：名称参数。
- opacity：透明度参数，范围是 0～1。
- rsrc、xsrc、ysrc、tsrc、idssrc、textsrc、textpositionsrc：字符串源数组列表，可作为 Plotly 网格标识符，用于设置特殊图表所需的 r 参数、x 参数、y 参数、t 参数、ids 参数、text（文本）参数和 textposition（文本位置）参数。
- r、t：仅用于极坐标图，r 用于设置径向坐标（半径），t 用于设置角坐标。
- showlegend：布尔变量，用于切换图标显示。
- stream：数据流，用于实时显示数据图表。
- textfont：文本字体参数，包括字体名称、颜色、大小等。
- textposition：“文本”元素的位置参数，包括 top left（左上）、top center（中上）、top right（右上）、middle left（左中）、middle center（中心）、middle right（右中）、bottom left（左下）、bottom center（中下）、bottom right（右下）模式，默认是 middle center（中心）模式。
- text：文本数据，设置与每个“（x，y）对”关联的文本元素和数组列表格式，默认是空字符串。
- type：数据显示模式，包括 constant（常数）、percent（百分比）、sqrt（平方根）、array（数组）模式。
- x0、y0：坐标轴起点坐标。
- xaxis，yaxis：*x*、*y* 坐标参数。
- xcalendar、ycalendar：坐标时间参数的格式，默认是公历（gregorian），支持 gregorian、chinese、coptic、discworld、ethiopian、hebrew、islamic、julian、mayan、nanakshahi、nepali、persian、jalali、taiwan、thai 和 ummalqura 格式。
- x，y：设置 *x*、*y* 轴的坐标数据。

2.4 气泡图

气泡图的实现方法与散点图的实现方法类似，修改散点图中的点的大小，就变成了气泡图。

2.4.1 基本案例

下面的代码（见文件 bubble_basic_demo.py）用来说明气泡图的基本绘制方法。

```
import plotly as py
import plotly.graph_objs as go

# ----------pre def
pyplt = py.offline.plot

# ----------code
trace0 = go.Scatter(
    x=[1, 2, 3, 4],
    y=[10, 11, 12, 13],
    mode='markers',
    marker=dict(
        size=[40, 60, 80, 100],
    )
)

data = [trace0]
pyplt(data, filename='tmp/bubble_baisc_demo.html')
```

代码运行结果如图 2-7 所示。

以上代码的核心代码如下，用来定义每一个点的大小：

```
marker=dict(
        size=[40, 60, 80, 100],
    )
```

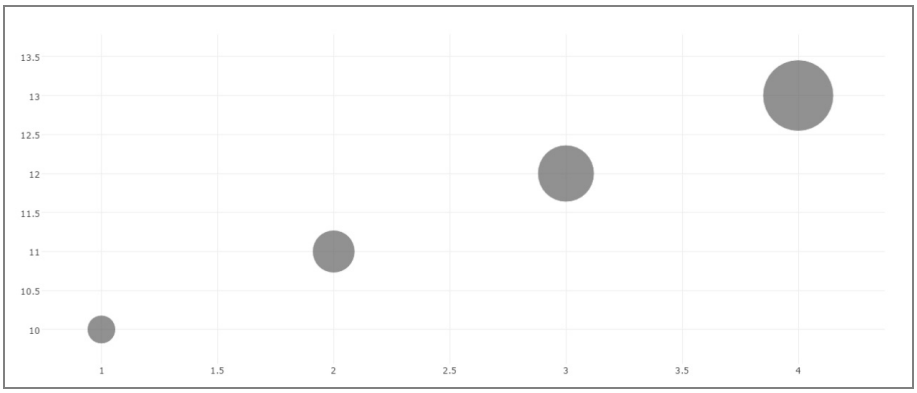

图 2-7　代码运行结果

2.4.2　样式设置

下面的代码（见文件 bubble_style.py）用来指定每个点的大小、颜色、悬浮文字和是否显示颜色条。

```
    import plotly as py
    import plotly.graph_objs as go

    # ----------pre def
    pyplt = py.offline.plot

    # ----------code
    trace0 = go.Scatter(
        x=[1, 2, 3, 4],
        y=[10, 11, 12, 13],
        mode='markers',
        text=['A<br>size: 40', 'B<br>size: 60', 'C<br>size: 80', 'D<br>size:
100'],
        marker=dict(
            color= [120, 125, 130, 135],
            opacity=[1, 0.8, 0.6, 0.4],
            size=[40, 60, 80, 100],
            showscale= True,
```

```
        )
    )

    data = [trace0]
    pyplt(data, filename='tmp/bubble_style.html')
```

代码运行结果如图 2-8 所示。

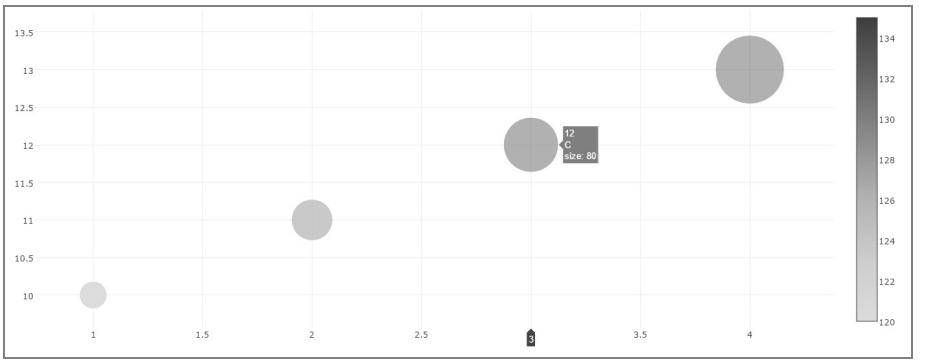

图 2-8　代码运行结果

以上代码的核心代码如下，text 指定每个点对应的悬浮文字（
表示换行），color 指定每个点的颜色，opacity 指定每个点的透明度，size 指定每个点的大小，showscale=True 表示显示右边的颜色条的大小。

```
    text=['A<br>size: 40', 'B<br>size: 60', 'C<br>size: 80', 'D<br>size:
100'],
    marker=dict(
           color=['rgb(93, 164, 214)', 'rgb(255, 144, 14)',
                  'rgb(44, 160, 101)', 'rgb(255, 65, 54)'],
           opacity=[1, 0.8, 0.6, 0.4],
           size=[40, 60, 80, 100],
           showscale= True,
       )
```

2.4.3　缩放设置

调节气泡尺寸的大小可通过 sizeref 参数进行设置，当 sizeref 值大于 1 时，将减

小气泡的大小；当 sizeref 值小于 1 时，将增大气泡的大小，详情参考页面 https://plot.ly/python/reference/#scatter-marker-sizeref 。本案例代码（见文件 bubble_scale.py）如下。

```
    import plotly as py
    import plotly.graph_objs as go

    # ----------pre def
    pyplt = py.offline.plot

    # ----------code
    trace0 = go.Scatter(
        x=[1, 2, 3, 4],
        y=[10, 11, 12, 13],
        text=['A</br>size: 40</br>default', 'B</br>size: 60</br>default',
'C</br>size: 80</br>default', 'D</br>size: 100</br>default'],
        mode='markers',
        name='default',
        marker=dict(
            size=[400, 600, 800, 1000],
            sizemode='area',
        )
    )
    trace1 = go.Scatter(
        x=[1, 2, 3, 4],
        y=[14, 15, 16, 17],
        text=['A</br>size: 40</br>sizeref: 0.2', 'B</br>size: 60</br>sizeref:
0.2', 'C</br>size: 80</br>sizeref: 0.2', 'D</br>size: 100</br>sizeref:
0.2'],
        mode='markers',
        name = 'ref0.2',
        marker=dict(
            size=[400, 600, 800, 1000],
            sizeref=0.2,
            sizemode='area',
        )
```

```
    )
    trace2 = go.Scatter(
        x=[1, 2, 3, 4],
        y=[20, 21, 22, 23],
        text=['A</br>size: 40</br>sizeref: 2', 'B</br>size: 60</br>sizeref:
2', 'C</br>size: 80</br>sizeref: 2', 'D</br>size: 100</br>sizeref: 2'],
        mode='markers',
        name='ref2',
        marker=dict(
            size=[400, 600, 800, 1000],
            sizeref=2,
            sizemode='area',
        )
    )

    data = [trace0, trace1, trace2]
    pyplt(data, filename='tmp/bubble_scale.html')
```

代码运行结果如图 2-9 所示。

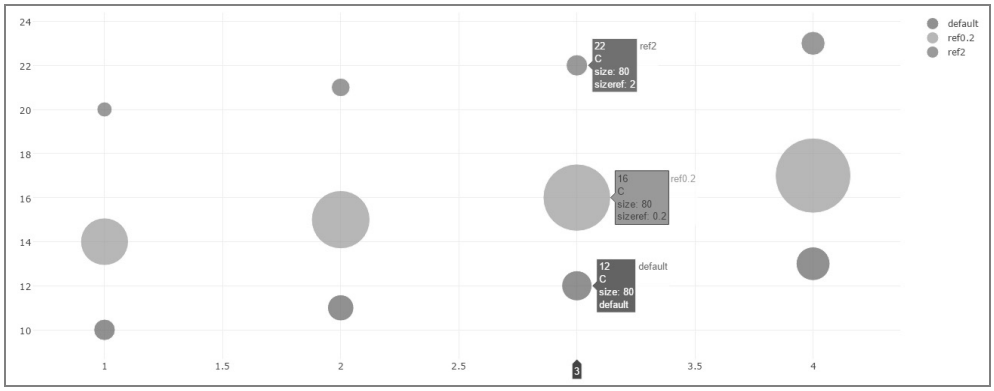

图 2-9　代码运行结果

与之前的代码相比，多了如下参数内容：参数 sizeref=2 表示将气泡大小设置为原来的 1/2；参数 sizemode 有 diameter 和 area 两个值，diameter 表示按直径缩放，area 表示按面积缩放，这里使用的是按面积缩放。

```
    marker=dict(
           size=[400, 600, 800, 1000],
           sizeref=2,
           sizemode='area',
        )
```

2.4.4　参数解读

气泡图与散点图使用的是同一个函数，因此大部分参数在 2.3.4 小节已经介绍过了，这里对本节所涉及的参数进行补充说明。

- text：列表，元素为相应节点的悬浮文字内容。
- marker：数据节点参数，包括大小、颜色、格式等，有如下设置项。
  - size：列表，元素为相应节点的尺寸大小。
  - sizeref：缩放的比例，如设置为 2，则缩小为原来的 1/2。
  - sizemode：缩放的标准，默认以 diameter（直径）缩放，也可选择以 area（面积）缩放。
  - color：列表，元素为相应节点的颜色。
  - showscale：默认为 False，不显示右侧的颜色条，也可以选择 True。
  - opacity：列表，元素为 0～1 之间的数，表示相应节点的透明度。

2.5　线形图

2.5.1　基本案例

线形图的绘制在散点图的绘制中提及过，用 Plotly 绘制线形图使用 Scatter 函数。如图 2-10 所示是线形图的简单实现，见文件 2.5_LineChart_1.py。本案例使用 Pandas 生成时间序列作为横轴标签，对浦发银行 2017 年 3 月 1 日—2017 年 4 月 28 日的股价涨跌幅进行了展现，数据来源是 Wind 数据库。

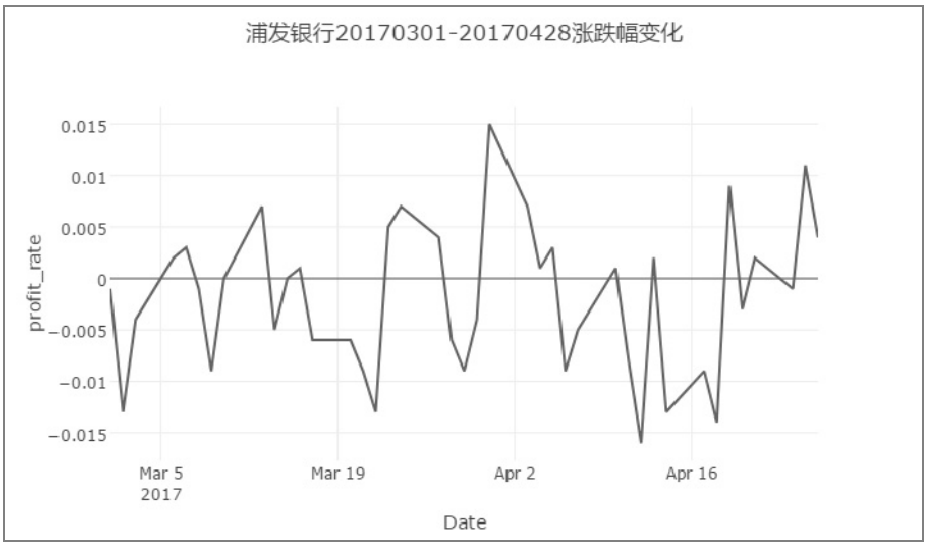

图 2-10　基本线形图

```
    # 2.5-1 基本案例
    import plotly as py
    import plotly.graph_objs as go

    # Basic Line
    pyplt = py.offline.plot
    # 600000 浦发银行 20170301-20170428 涨跌幅度数据，数据来源 Wind
    profit_rate = [-0.001, -0.013, -0.004, 0.002, 0.003, -0.001, -0.009, 0.0,\
0.007,
    -0.005, 0.0, 0.001, -0.006, -0.006, -0.009, -0.013, 0.005, 0.007,\
        0.004, -0.006, -0.009, -0.004, 0.015, 0.007, 0.001, 0.003, -0.009,\
        -0.005, 0.001, -0.008, -0.016, 0.002, -0.013, -0.009, -0.014, 0.009,\
        -0.003, 0.002, -0.001, 0.011, 0.004]
    date = pd.date_range(start = '3/1/2017', end = '4/30/2017')
    trace = [go.Scatter(
        x = date,
        y = profit_rate
    )]
    layout = dict(
```

```
    title = '浦发银行 20170301-20170428 涨跌幅变化',
    xaxis = dict(title = 'Date'),
    yaxis = dict(title = 'profit_rate')
)

fig = dict(data = trace, layout = layout)
pyplt(fig, filename='tmp/basic-line.html')
```

2.5.2　数据缺口与连接

在实际应用过程中，数据集往往并不完美，可能有缺失的数据，在 Plotly 中可以通过设置 Scatter 函数中的 connectgaps 属性来显示这些数据缺口或对缺口进行连接。如图 2-11 所示是在官方案例的基础上进行的调整，包含了多条线形图的绘制、线条样式设置，以及数据缺口保留与连接的控制，见文件 2.5_LineChart_2.py。

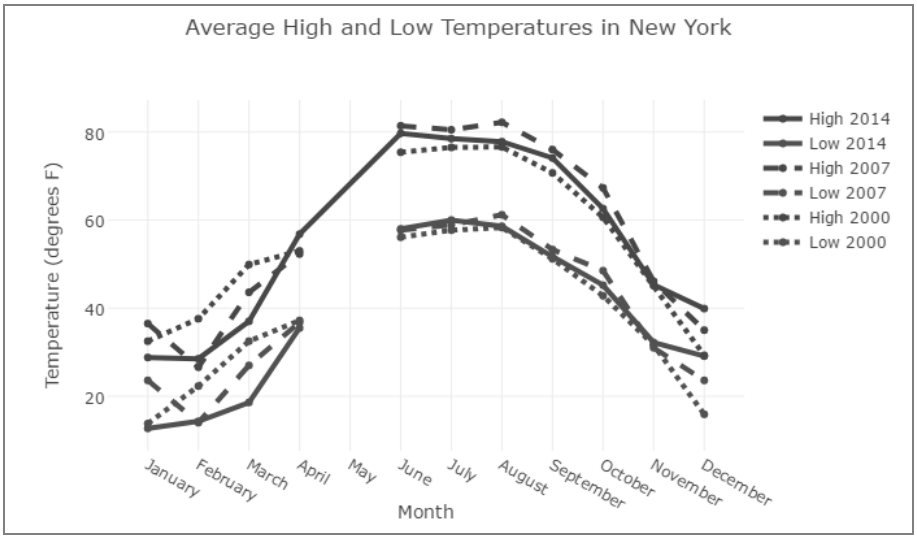

图 2-11　线形图缺失数据展示与连接

该案例的代码如下。

```
# 2.5-2 应用案例
# Average High and Low Temperatures in New York
```

```
    import plotly as py
    import plotly.graph_objs as go
    pyplt = py.offline.plot
    month = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
            'August', 'September', 'October', 'November', 'December'] # x
轴坐标
    high_2000 = [32.5, 37.6, 49.9, 53.0, None, 75.4, 76.5, 76.6, 70.7, 60.6,
45.1, 29.3]
    low_2000 = [13.8, 22.3, 32.5, 37.2, None, 56.1, 57.7, 58.3, 51.2, 42.8,
31.6, 15.9]
    high_2007 = [36.5, 26.6, 43.6, 52.3, None, 81.4, 80.5, 82.2, 76.0, 67.3,
46.1, 35.0]
    low_2007 = [23.6, 14.0, 27.0, 36.8, None, 57.7, 58.9, 61.2, 53.3, 48.5,
31.0, 23.6]
    high_2014 = [28.8, 28.5, 37.0, 56.8, None, 79.7, 78.5, 77.8, 74.1, 62.6,
45.3, 39.9]
    low_2014 = [12.7, 14.3, 18.6, 35.5, None, 58.0, 60.0, 58.6, 51.7, 45.2,
32.2, 29.1]
    # 6 组数据
    # Create and style traces
    trace0 = go.Scatter(
        x = month,
        y = high_2014,
        name = 'High 2014',
        line = dict(
            color = ('rgb(205, 12, 24)'),
            width = 4),
        connectgaps = True
    )
    trace1 = go.Scatter(
        x = month,
        y = low_2014,
        name = 'Low 2014',
        line = dict(
            color = ('rgb(22, 96, 167)'),
            width = 4,),
        connectgaps = False
```

```
)
trace2 = go.Scatter(
    x = month,
    y = high_2007,
    name = 'High 2007',
    line = dict(
        color = ('rgb(205, 12, 24)'),
        width = 4,
        dash = 'dash'),
    connectgaps = False
)
# dash 虚线（短线），dot 虚线（点），dashdot
trace3 = go.Scatter(
    x = month,
    y = low_2007,
    name = 'Low 2007',
    line = dict(
        color = ('rgb(22, 96, 167)'),
        width = 4,
        dash = 'dash'),
    connectgaps = False
)
trace4 = go.Scatter(
    x = month,
    y = high_2000,
    name = 'High 2000',
    line = dict(
        color = ('rgb(205, 12, 24)'),
        width = 4,
        dash = 'dot'),
    connectgaps = False
)
trace5 = go.Scatter(
    x = month,
    y = low_2000,
    name = 'Low 2000',
```

```
        line = dict(
            color = ('rgb(22, 96, 167)'),
            width = 4,
            dash = 'dot'),
        connectgaps = False
    )
    data = [trace0, trace1, trace2, trace3, trace4, trace5]

    # Edit the layout
    layout = dict(title = 'Average High and Low Temperatures in New York',
                  xaxis = dict(title = 'Month'),
                  yaxis = dict(title = 'Temperature (degrees F)'),
                  )

    fig = dict(data=data, layout=layout)
    pyplt(fig, filename='tmp/styled-line.html')
```

在数据部分，原先的缺失数据被设置为 None。在 Scatter 函数中，设置 connectgaps 属性为 Fasle，表示不连接，显示数据缺口；设置 connectgaps 属性为 True，表示连接缺失值左右相邻的数据点。在图 2-11 中，对“High 2014”线形图进行了连接，其他线条则采用显示缺口的形式。

Scatter 函数中的 line 属性用于对线形图的样式进行控制；color 用于设置颜色；width 用于设置宽度；dash 用于设置类型，dash 表示由短线组成的虚线，dot 表示由点组成的虚线，dashdot 表示由点和短线组成的虚线。

2.5.3 数据插值

通过调整 Scatter 函数 line 属性中的 shape 值可以对插值的方法进行控制，完成数据点的插值设置。插值的方法简单来说就是根据已有的零散数据点，找到一条满足一定条件的曲线，使之经过全部的数据点。Plotly 提供的插值方法有 6 种，分别是'linear'、'spline'、'hv'、'vh'、'hvh'和'vhv'。例如，设置 shape='spline'，表示通过三次样条方法对数据点进行插值。图 2-12 所示为官方案例，展示了 6 种不同的插值方法，见文件 2.5_LineChart_3.py。

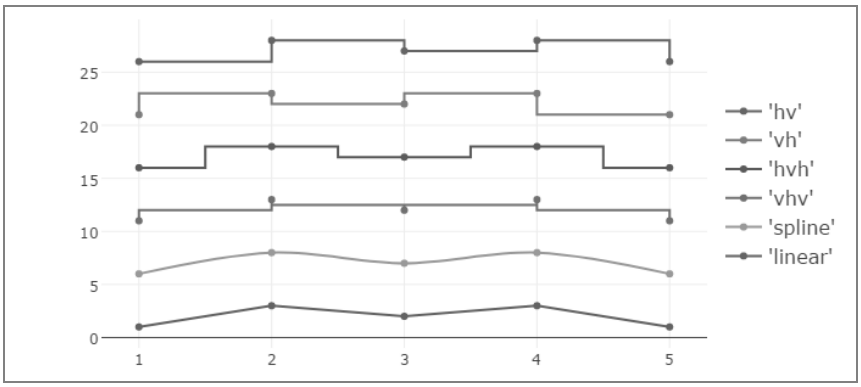

图 2-12　不同插值方法的对比

该案例的代码如下。

```
# 2.5-3 应用案例
import plotly as py
import plotly.graph_objs as go
pyplt = py.offline.plot
trace1 = go.Scatter(
    x=[1, 2, 3, 4, 5],
    y=[1, 3, 2, 3, 1],
    mode='lines+markers',
    name="'linear'",
    hoverinfo='name',
    line=dict(
        shape='linear'
    )
)
trace2 = go.Scatter(
    x=[1, 2, 3, 4, 5],
    y=[6, 8, 7, 8, 6],
    mode='lines+markers',
    name="'spline'",
    text=["tweak line smoothness<br>with 'smoothing' in line object"],
    hoverinfo='text+name',
    line=dict(
        shape='spline'
    )
)
```

```
trace3 = go.Scatter(
    x=[1, 2, 3, 4, 5],
    y=[11, 13, 12, 13, 11],
    mode='lines+markers',
    name="'vhv'",
    hoverinfo='name',
    line=dict(
        shape='vhv'
    )
)
trace4 = go.Scatter(
    x=[1, 2, 3, 4, 5],
    y=[16, 18, 17, 18, 16],
    mode='lines+markers',
    name="'hvh'",
    hoverinfo='name',
    line=dict(
        shape='hvh'
    )
)
trace5 = go.Scatter(
    x=[1, 2, 3, 4, 5],
    y=[21, 23, 22, 23, 21],
    mode='lines+markers',
    name="'vh'",
    hoverinfo='name',
    line=dict(
        shape='vh'
    )
)
trace6 = go.Scatter(
    x=[1, 2, 3, 4, 5],
    y=[26, 28, 27, 28, 26],
    mode='lines+markers',
    name="'hv'",
    hoverinfo='name',
    line=dict(
```

```
        shape='hv'
    )
)
data = [trace1, trace2, trace3, trace4, trace5, trace6]
layout = dict(
    legend=dict(
        y=0.5,
        traceorder='reversed',
        font=dict(
            size=16
        )
    )
)
fig = dict(data=data, layout=layout)
pyplt(fig, filename='tmp/line-shapes.html')
```

2.5.4　填充线形图

填充线形图是线形图的一种衍生，通过选择性地显示线条和对线条图进行填充来完成。如图 2-13 所示展示了恒宝股份、湘潭电化、大港股份的股票在一段时期内开盘的最高价与最低价，每条可见线条对应股票的开盘价，线条的上影线对应当天的最高价，线条的下影线对应当天的最低价，见文件 2.5_LineChart_4.py。

要绘制这样一个可视化图形，先把其拆成两部分，一部分是对三条可见线条（开盘价线条）进行绘制，另一部分是对三条填充线条进行绘制。下面这段代码完成了这个操作。

```
x = x + x_rev,
y = y1_upper + y1_lower,
fill = 'tozerox',
fillcolor = 'rgba(0,0,205,0.2)',
line = go.Line(color = 'transparent'),
```

首先，x + x_rev 是从 1 到 10、再从 10 到 1 的序列，y1_upper + y1_lower 是从第 1 天的最高价至第 10 天的最高价、再从第 10 天的最高价至第 1 天的最高价的序列，注意这里的 y1_lower 已经在数据设置部分设置为逆序，由此可以得到两条线，

通过对 fill 属性的设置，即可对两条线之间的部分进行颜色填充，最后设置 line 中的 color 属性为'transparent'，对线条进行隐藏，运行结果如图 2-13 所示。

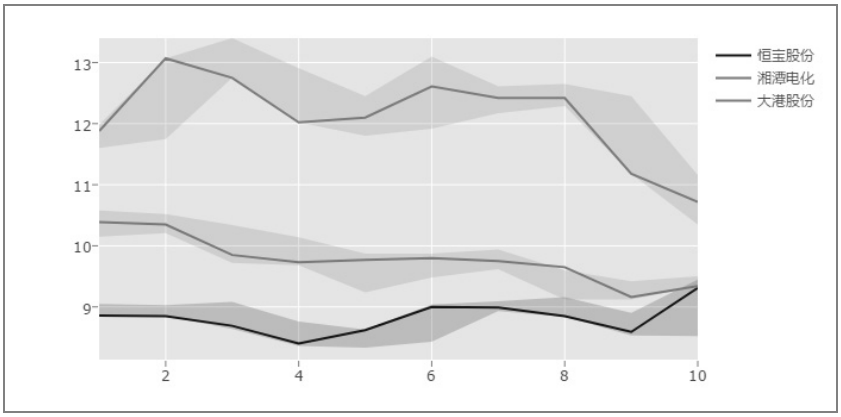

图 2-13　填充线形图

该案例的代码如下。

```
# 2.5-4 应用案例
import plotly as py
import plotly.graph_objs as go
pyplt = py.offline.plot
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x_rev = x[::-1]

# Line 1 002104 恒宝股份 20170518-20170602
y1 = [8.86, 8.85, 8.69, 8.4, 8.62, 9, 8.99, 8.85, 8.59, 9.31]
y1_upper = [9.05, 9.03, 9.08, 8.76, 8.63, 9.04, 9.09, 9.16, 8.9, 9.45]
y1_lower = [8.86, 8.85, 8.64, 8.36, 8.33, 8.43, 8.93, 8.84, 8.53, 8.52]
y1_lower = y1_lower[::-1] # 逆序

# Line 2 002125 湘潭电化 20170518-20170602
y2 = [10.39, 10.35, 9.85, 9.73, 9.77, 9.8, 9.75, 9.65, 9.16, 9.34]
y2_upper = [10.58, 10.52, 10.34, 10.14, 9.87, 9.87, 9.94, 9.6, 9.42, 9.5]
y2_lower = [10.15, 10.21, 9.72, 9.68, 9.24, 9.48, 9.62, 9.12, 9.12, 9.34]
y2_lower = y2_lower[::-1]

# Line 3 002077 大港股份 20170518-20170602
```

```
    y3 = [11.88, 13.07, 12.75, 12.02, 12.1, 12.61, 12.42, 12.42, 11.18, 10.72]
    y3_upper = [11.98, 13.07, 13.4, 12.91, 12.45, 13.1, 12.61, 12.65, 12.45,
11.16]
    y3_lower = [11.6, 11.75, 12.75, 12.02, 11.8, 11.92, 12.17, 12.29, 11.18,
10.35]
    y3_lower = y3_lower[::-1]

    trace1 = go.Scatter(
        x = x + x_rev,
        y = y1_upper + y1_lower,
        fill = 'tozerox',
        fillcolor = 'rgba(0,0,205,0.2)',
        line = go.Line(color = 'transparent'),
        showlegend = False,
        name = '恒宝股份',
    )
    trace2 = go.Scatter(
        x = x + x_rev,
        y = y2_upper + y2_lower,
        fill = 'tozerox',
        fillcolor = 'rgba(30,144,255,0.2)',
        line = go.Line(color = 'transparent'),
        name = '湘潭电化',
        showlegend = False,
    )
    trace3 = go.Scatter(
        x = x+x_rev,
        y = y3_upper+y3_lower,
        fill = 'tozerox',
        fillcolor = 'rgba(112,128,144,0.2)',
        line = go.Line(color = 'transparent'),
        showlegend = False,
        name = '大港股份',
    )
    trace4 = go.Scatter(
        x = x,
        y = y1,
```

```
    line = go.Line(color = 'rgb(0,0,205)'),
    mode = 'lines',
    name = '恒宝股份',
)
trace5 = go.Scatter(
    x = x,
    y = y2,
    line = go.Line(color='rgb(30,144,255)'),
    mode = 'lines',
    name = '湘潭电化',
)
trace6 = go.Scatter(
    x = x,
    y = y3,
    line = go.Line(color='rgb(112,128,144)'),
    mode = 'lines',
    name = '大港股份',
)

data = go.Data([trace1, trace2, trace3, trace4, trace5, trace6])

layout = go.Layout(
    paper_bgcolor = 'rgb(255,255,255)',
    plot_bgcolor = 'rgb(229,229,229)',
    xaxis = go.XAxis(
        gridcolor = 'rgb(255,255,255)',
        range = [1,10],
        showgrid = True,
        showline = False,
        showticklabels = True,
        tickcolor = 'rgb(127,127,127)',
        ticks = 'outside',
        zeroline = False
    ),
    yaxis = go.YAxis(
        gridcolor = 'rgb(255,255,255)',
        showgrid = True,
```

```
        showline = False,
        showticklabels = True,
        tickcolor = 'rgb(127,127,127)',
        ticks = 'outside',
        zeroline = False
    ),
)
fig = go.Figure(data = data, layout = layout)
pyplt(fig, filename = 'tmp/shaded_lines.html')
```

2.5.5　应用案例

新闻来源统计线形图案例的运行结果如图 2-14 所示，代码见文件 2.5_LineChart_5.py。

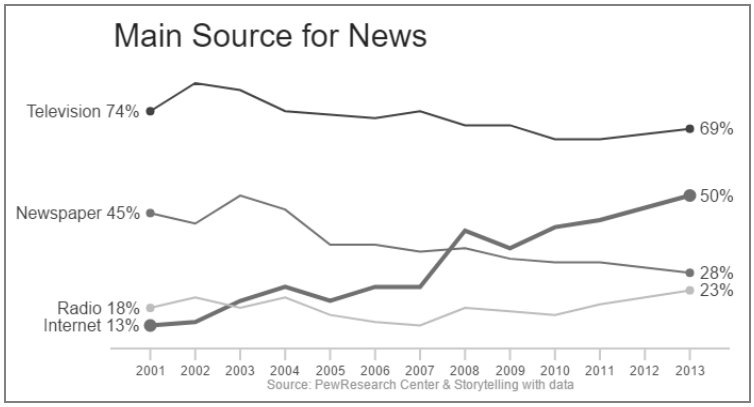

图 2-14　新闻来源统计线形图

该案例的代码如下。

```
# 2.5-5 应用案例
import plotly as py
import plotly.graph_objs as go
pyplt = py.offline.plot
title = 'Main Source for News'

labels = ['Television', 'Newspaper', 'Internet', 'Radio']

colors = ['rgba(67,67,67,1)', 'rgba(115,115,115,1)', 'rgba(49,130,189,
```

```
1)', 'rgba(189,189,189,1)']

    mode_size = [8, 8, 12, 8]

    line_size = [2, 2, 4, 2]

    x_data = [
       [2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011,
2013],
       [2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011,
2013],
       [2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011,
2013],
       [2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011,
2013],
    ]

    y_data = [
       [74, 82, 80, 74, 73, 72, 74, 70, 70, 66, 66, 69],
       [45, 42, 50, 46, 36, 36, 34, 35, 32, 31, 31, 28],
       [13, 14, 20, 24, 20, 24, 24, 40, 35, 41, 43, 50],
       [18, 21, 18, 21, 16, 14, 13, 18, 17, 16, 19, 23],
    ]

    traces = []

    for i in range(0, 4):
       traces.append(go.Scatter(
          x = x_data[i],
          y = y_data[i],
          mode = 'lines',
          line = dict(color = colors[i], width = line_size[i]),
          connectgaps = True,
       ))

       traces.append(go.Scatter(
          x = [x_data[i][0], x_data[i][11]],
          y = [y_data[i][0], y_data[i][11]],
          mode = 'markers',
          marker = dict(color = colors[i], size = mode_size[i])
```

```
    ))

layout = go.Layout(
    xaxis = dict(
        showline = True,
        showgrid = False,
        showticklabels = True, # True 为显示坐标标记
        linecolor = 'rgb(204, 204, 204)',# x 轴线的颜色
        linewidth = 2,
        autotick = False,  # True 为自动删除部分日期，False 为保持原状
        ticks = 'outside', # x 轴上的刻度线，在图内或图外
        tickcolor = 'rgb(204, 204, 204)', # x 轴上的刻度线的颜色
        tickwidth = 2, # x 轴上的刻度线的宽度
        ticklen = 10,  # x 轴上的刻度线的长度
        tickfont=dict(        # x 轴上的坐标标记的字体样式、大小、颜色
            family = 'Arial',
            size = 12,
            color = 'rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        showgrid = False,
        zeroline = False,
        showline = False,
        showticklabels = False,
    ),
    autosize = False,
    margin = dict(
        autoexpand = False,
        l = 100,
        r = 20,
        t = 110,
    ),
    showlegend = False,
)

annotations = []

# Adding labels
```

```
    for y_trace, label, color in zip(y_data, labels, colors):
        # labeling the left_side of the plot
        annotations.append(dict(xref = 'paper', x = 0.05, y = y_trace[0],
                                xanchor = 'right', yanchor = 'middle',
                                text = label + ' {}%'.format(y_trace[0]),
                                font = dict(family = 'Arial',
                                        size = 16,
                                        color = colors,),
                                showarrow = False))
        # labeling the right_side of the plot
        annotations.append(dict(xref = 'paper', x = 0.95, y = y_trace[11],
                                xanchor = 'left', yanchor = 'middle',
                                text = '{}%'.format(y_trace[11]),
                                font = dict(family = 'Arial',
                                        size = 16,
                                        color = colors,),
                                showarrow = False))
    # Title
    annotations.append(dict(xref = 'paper', yref = 'paper', x = 0.0, y = 1.05,
                            xanchor = 'left', yanchor = 'bottom',
                            text = 'Main Source for News',
                            font = dict(family = 'Arial',
                                    size = 30,
                                    color = 'rgb(37,37,37)'),
                            showarrow = False))
    # Source
    annotations.append(dict(xref = 'paper', yref = 'paper', x = 0.5, y = -0.1,
                            xanchor = 'center', yanchor = 'top',
                            text = 'Source: PewResearch Center & ' +
                                'Storytelling with data',
                            font = dict(family = 'Arial',
                                    size = 12,
                                    color = 'rgb(150,150,150)'),
                            showarrow = False))

    layout['annotations'] = annotations

    fig = go.Figure(data = traces, layout = layout)
    pyplt(fig, filename = 'tmp/news-source.html')
```

2.5.6　参数解读

由于线条图的绘制方法与散点图的绘制方法是一样的，都使用 Scatter 函数，所以它们的参数也是一样的，读者可以参考 2.3.4 节的相关内容。

2.6　柱状图

2.6.1　基本柱状图

使用 Plotly 绘制基本柱状图最重要的函数是 graph_objs 中的 Bar 函数，通过传递数据，可以设置柱状图的样式。在 Layout 中对 barmode 进行设置可以绘制出不同类型的柱状图。如图 2-15 所示是柱状图最简单的实现，见文件 2.6_BarChart_1.py，该案例包括使用 Bar 函数传递数据和变量，以及在 Layout 中设置标题、*x* 轴范围。

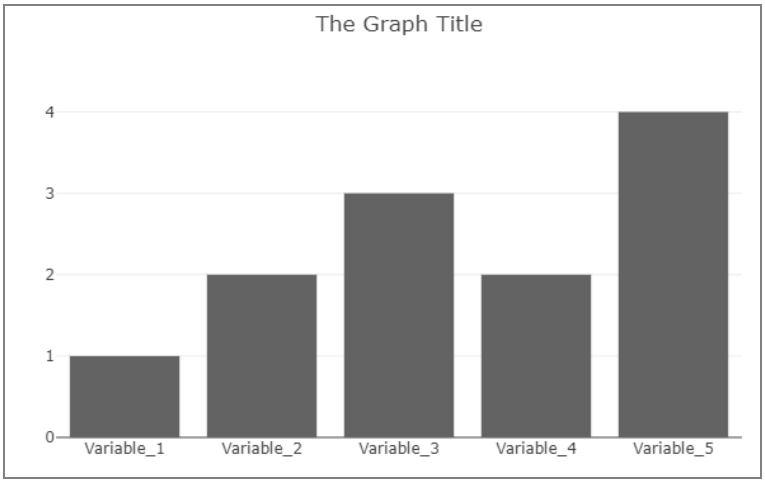

图 2-15　基本柱状图

```
# 2.6-1 基本案例
import plotly as py
import plotly.graph_objs as go
pyplt = py.offline.plot
```

```
    # Trace
    trace_basic = [go.Bar(
              x = ['Variable_1', 'Variable_2', 'Variable_3','Variable_4',
'Variable_5'],
              y = [1, 2, 3, 2, 4],
       )]

    # Layout
    layout_basic = go.Layout(
              title = 'The Graph Title',
              xaxis = go.XAxis(range = [-0.5,4.5], domain = [0,1])
       )

    # Figure
    figure_basic = go.Figure(data = trace_basic, layout = layout_basic)

    # Plot
    pyplt(figure_basic, filename='tmp/Basic_BarChart.html')
```

2.6.2 柱状簇

除基本柱状图外，还有柱状簇，在基本柱状图代码的基础上加入多组数据即可实现柱状簇。图 2-16 所示为国际贸易板块中广东明珠、五矿发展与上海物贸这 3 只股票从 2016 年第三季度到 2017 年第一季度的净资产收益率（%）变化，数据来源是同花顺。

本案例见文件 2.6_BarChart_2.py。从纵向看，上海物贸三个季度的数据分别为 4.12、3.65 和 2.15；从横向看，data1 中的数据 y = [4.12, 5.32, 0.60]是 3 只不同股票对应的资产收益率，通过这种配置数据的方式可以对柱状簇图进行实现。本案例通过柱状簇图对比了 2016 年第三季度、2016 年第四季度与 2017 年第一季度 3 个不同时间段同一只国际贸易板块股票的资产收益率，而且对同一时间段不同股票的资产收益率的差异也进行了展示。

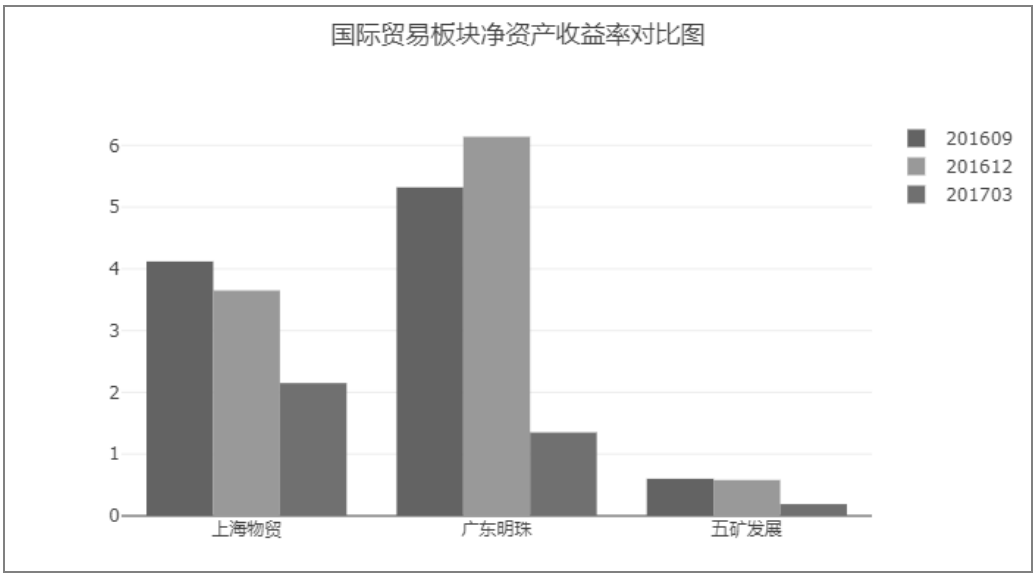

图 2-16　国际贸易板块净资产收益率的对比

本案例的代码如下。

```
# 2.6-2 应用案例
import plotly as py
import plotly.graph_objs as go
pyplt = py.offline.plot
# Traces
trace_1 = go.Bar(
          x = ["上海物贸", "广东明珠", "五矿发展"],
          y = [4.12, 5.32, 0.60],
          name = "201609"
    )

trace_2 = go.Bar(
          x = ["上海物贸", "广东明珠", "五矿发展"],
          y = [3.65, 6.14, 0.58],
          name = "201612"
    )
```

```
trace_3 = go.Bar(
          x = ["上海物贸", "广东明珠", "五矿发展"],
          y = [2.15, 1.35, 0.19],
          name = "201703"
    )

trace = [trace_1, trace_2, trace_3]

# Layout
layout = go.Layout(
          title = '国际贸易板块净资产收益率对比图'
    )

# Figure
figure = go.Figure(data = trace, layout = layout)

# Plot
pyplt(figure, filename='tmp/BarChart_1.html')
```

2.6.3 层叠柱状图

层叠柱状图的绘制与柱状簇图的绘制大同小异，相当于对同一簇的柱状图进行叠加，实现的方式是对 Layout 中的 barmode 属性进行设置，即设置 barmode='stack'，其余参数与柱状簇图相同。图 2-17 所示的层叠柱状图展示了华夏新经济混合、华夏上证 50、嘉实新机遇混合、南方消费活力混合和华泰柏瑞这 5 只基金的资产配置比例，数据来源是和讯网。

该案例见文件 2.6_BarChart_3.py，通过设置 barmode 属性实现柱状图的层叠。通过层叠柱状图可以很清楚地看出不同基金的资产配置情况，如华夏新经济混合、华夏上证 50、华泰柏瑞这 3 只基金有很大的股票投资权重。通过观察同一只基金的资产配置比例，可以对其风险属性有更具体的了解。

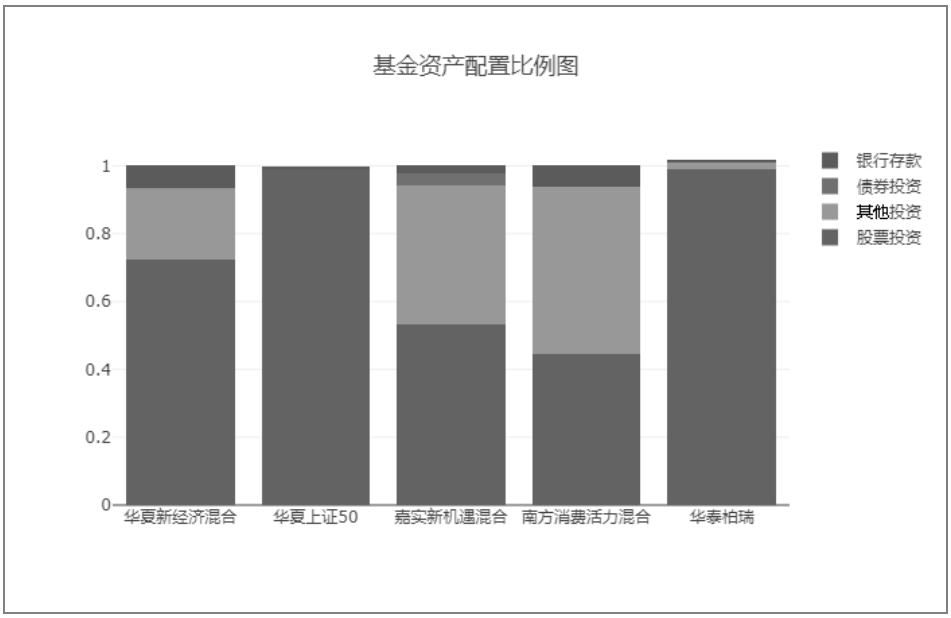

图 2-17　基金资产配置比例图

该案例的代码如下。

```
    # 2.6-3 应用案例
    import plotly.plotly as py
    import plotly.graph_objs as go
    pyplt = py.offline.plot

    # Data
    data_1 = go.Bar(
        x = ['华夏新经济混合', '华夏上证 50', '嘉实新机遇混合', '南方消费活力混合','
华泰柏瑞'],
        y = [0.7252, 0.9912, 0.5347, 0.4436, 0.9911],
        name = '股票投资'
    )

    data_2 = go.Bar(
        x = ['华夏新经济混合', '华夏上证 50', '嘉实新机遇混合', '南方消费活力混合','
华泰柏瑞'],
        y = [0.2072, 0, 0.4081, 0.4955, 0.02],
```

```
        name='其他投资'
    )

    data_3 = go.Bar(
        x = ['华夏新经济混合', '华夏上证 50', '嘉实新机遇混合', '南方消费活力混合','
华泰柏瑞'],
        y = [0, 0, 0.037, 0, 0],
        name='债券投资'
    )

    data_4 = go.Bar(
        x = ['华夏新经济混合', '华夏上证 50', '嘉实新机遇混合', '南方消费活力混合','
华泰柏瑞'],
        y = [0.0676, 0.0087, 0.0202, 0.0609, 0.0087],
        name='银行存款'
    )

    data = [data_1, data_2, data_3, data_4]

    # Layout
    layout = go.Layout(
        title = '基金资产配置比例图',
        barmode='stack'
    )
    # Figure
    fig = go.Figure(data=data, layout=layout)
    py.iplot(fig, filename='tmp/stacked-bar')
```

2.6.4 瀑布式柱状图

瀑布式柱状图是层叠柱状图的一种衍生，通过选择性地显示层叠部分来实现柱状图的悬浮效果。如图 2-18 所示的瀑布式柱状图展示了万科 A 股票在 2016 年的资产负债结构，其中非流动负债与所有者权益柱状图的悬浮效果通过设置 trace0 中的 *y* 值实现，如 *y*=[0, 57999848, 0, 66899764, 0]，表示第 1、3、5 根柱状图从 0 开始显示，第 2 根柱状图从 57999848 开始显示，第 4 根柱状图从 66899764 开始显示，实

现的方式是将 trace1 所表示的柱形颜色设置为白色，即 color='rgb(255, 255, 255)'，起到选择性显示的效果，见文件 2.6_BarChart_4.py。

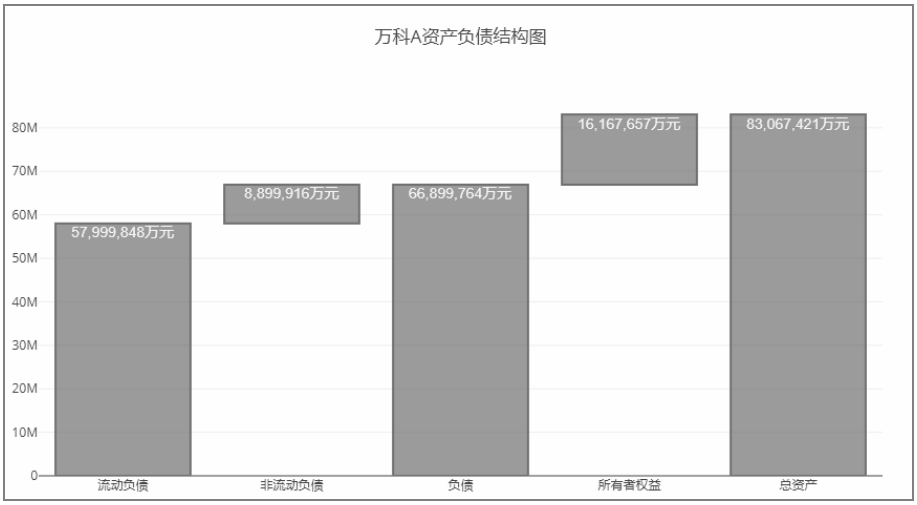

图 2-18　万科 A 股票资产负债结构

该案例的代码如下。

```
    # 2.6-4 应用案例
    import plotly as py
    import plotly.graph_objs as go
    pyplt = py.offline.plot

    x_data = ['流动负债', '非流动负债',
              '负债','所有者权益', '总资产']
    y_data = [56000000, 65000000, 65000000, 81000000, 81000000]
    text = ['57,999,848 万元', '8,899,916 万元', '66,899,764 万元', '16,167,657
万元', '83,067,421 万元']

    # Base
    trace0 = go.Bar(
        x=x_data,
        y=[0, 57999848, 0, 66899764, 0],
        marker=dict(
            color='rgb(255, 255, 255)',
        )
```

```
)
# Trace
trace1 = go.Bar(
    x=x_data,
    y=[57999848, 8899916, 66899764,16167657, 83067421],
    marker=dict(
        color='rgba(55, 128, 191, 0.7)',
        line=dict(
            color='rgba(55, 128, 191, 1.0)',
            width=2,
        )
    )
)

data = [trace0, trace1]
layout = go.Layout(
    title = '万科A资产负债结构图',
    barmode='stack',
    showlegend=False
)

annotations = []

for i in range(0, 5):
    annotations.append(dict(x=x_data[i], y=y_data[i], text=text[i],
                            font=dict(family='Arial', size=14,
                            color='rgba(245, 246, 249, 1)'),
                            showarrow=False,))
    layout['annotations'] = annotations

fig = go.Figure(data=data, layout=layout)
pyplt(fig, filename = 'tmp/waterfall-bar-profit.html')
```

2.6.5 图形样式设置

对柱状图颜色与样式的设置通过下面这个案例来说明。图 2-19 所示的柱状图展

示了有色金属板块 AU、AG、SN、PB 和 CU 这 5 种合约在某一个交易日的最高涨幅与波动率，柱形的宽度表示相对波动率的高低，柱形越宽，波动率越大；高度表示涨幅，红色柱状图突出了涨幅上升的商品期货，绿色柱状图突出了涨幅下跌的商品期货。

本案例见文件 2.6_BarChart_5.py，包括设置柱状图的颜色样式、用 line 设置柱状图外围的框线、用 width 设置柱状图的宽度、用 opacity 设置柱状图颜色的透明度，以及设置 Layout 中的 xaxis 令 *x* 轴的标记旋转–45°。

图 2-19　有色金属板块主力合约日内最高涨幅与波动率图

本案例的代码如下。

```
# 2.6-5 应用案例
import plotly as py
import plotly.graph_objs as go
pyplt = py.offline.plot

# Customizing Individual Bar Colors
volume = [0.49,0.71,1.43,1.4,0.93]
width = [each*3/sum(volume) for each in volume]
```

```
trace0 = go.Bar(
    x = ['AU.SHF', 'AG.SHF', 'SN.SHF',
       'PB.SHF', 'CU.SHF'],
    y = [0.85, 0.13, -0.93, 0.46, 0.06],
    width = width,
    marker = dict(
        color=['rgb(205,38,38)', 'rgb(205,38,38)',
              'rgb(34,139,34)', 'rgb(205,38,38)',
              'rgb(205,38,38)'],
        line=dict(
            color='rgb(0,0,0)',
            width=1.5,
        )),
        opacity = 0.8,
)

data = [trace0]
layout = go.Layout(
    title = '有色金属板块主力合约日内最高涨幅与波动率图',
    xaxis=dict(tickangle=-45),
)

fig = go.Figure(data=data, layout=layout)
pyplt(fig, filename='tmp/highlight-bar.html')
```

2.6.6 应用案例

经过对前面案例的学习，读者对柱状图的绘制已经很清楚了，在实际运用时，往往需要从 dataframe 中获取高频金融数据进行可视化，下面的案例讲解从导入 CSV 文件到做出成交量柱状图的过程，更加贴近现实中的应用，运行结果如图 2-20 所示，见文件 2.6_BarChart_6.py。

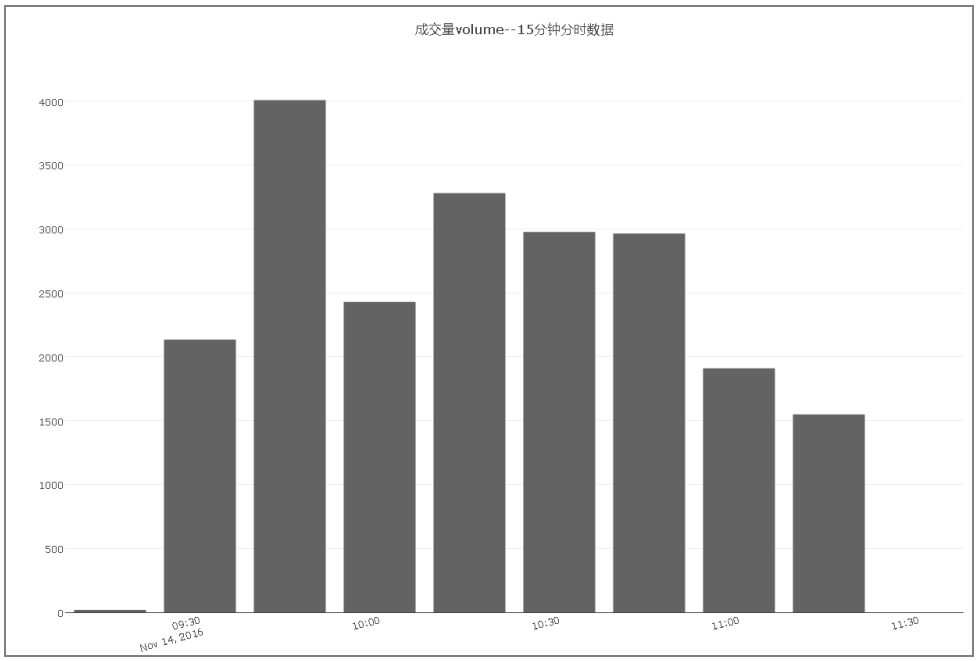

图 2-20　成交量柱状图

本案例的代码如下。

```
# 2.6-6 应用案例
import plotly as py
import plotly.graph_objs as go

pd.set_option('display.width', 450)
pyplt=py.offline.plot
df=pd.read_csv('dat/tk01_m15.csv')
df2= df[:10];print(df9)
xtr = go.Bar(
    x=df2['time'],
    y=df2['volume'],

)
xdat = go.Data([xtr])
layout = go.Layout(
    title = '成交量 volume--15 分钟分时数据',
```

```
        xaxis = go.XAxis(tickangle=-15),
    )
    fig = go.Figure(data=xdat, layout=layout)
    pyplt(fig)
```

2.6.7 参数解读

本小节对绘制柱状图所需 Bar 函数的常用参数进行详细讲解，包括设置柱状图所需的数据和属性，常用参数如下。

- base：柱状图的起始参数。
- dx、dy：*x*、*y* 坐标轴的步进值，默认值是 1。
- error_x、error_y：*x*、*y* 出错信息。
- hoverinfo：当用户与图形互动时，鼠标指针显示的参数，包括 *x*、*y*、*z* 坐标数据，以及 text（文字信息）和 name（图形名称）数据等参数的组合，使用+、all、none 和 skip（忽略）作为组合连接符号，默认是 all（全部显示）。
- insidetextfont：内置文本的字体格式参数。
- legendgroup：图标参数，默认是空字符串。
- marker：数据节点参数，包括大小、颜色、格式等。
- name：名称参数。
- offset：坐标位移参数。
- opacity：透明度参数，范围是 0～1。
- orientation：图形显示方向参数，包括 v（垂直模式）和 h（水平模式）。
- outsidetextfont：外置文本的字体参数。
- rsrc、xsrc、ysrc、textsrc、textpositionsrc、offsetsrc、basesrc、widthsrc：字符串源数组列表，作为 Plotly 网格标识符，用于设置一些特殊图表所需的 r 参数、x 参数、y 参数、text（文本）参数、textposition（文本位置）参数、offset（位移）参数、base（起点）参数、width（宽度）参数。
- r、t：仅用于极坐标图，r 用于设置径向坐标（半径），t 用于设置角坐标。
- showlegend：布尔变量，用于切换图标显示。
- stream：数据流，用于实时同步数据图表。
- textfont：文本字体参数，包括字体名称、颜色、大小等。

- textposition：“文本”元素的位置参数，包括 top left（左上）、top center（中上）、top right（右上）、middle left（左中）、middle center（中心）、middle right（右中）、bottom left（左下）、bottom center（中下）、bottom right（右下）模式。默认是 middle center（中心）模式。
- text：文本数据，设置与每个“(x,y）对”关联的文本元素，数组列表格式，默认是空字符串。
- type：数据显示模式参数，包括 constant（常数）、percent（百分比）、sqrt（平方根）和 array（数组）。
- visible：布尔变量，切换图形显示开关。
- width：柱状图的条形宽度。
- x0、y0：坐标轴起点坐标。
- xaxis、yaxis：*x*、*y* 坐标参数。
- xcalendar、ycalendar：坐标时间参数格式，默认是公历（gregorian）。
- x、y：　*x*、*y* 轴的坐标数据。

2.7　水平条形图

2.7.1　基本案例

使用 Plotly 绘制水平条形图与绘制柱状图类似，只是需要在 Bar 函数中设置 orientation = 'h'，其余参数与柱状图相同，也可以通过设置 barmode = 'stack'绘制层叠水平条形图与瀑布式水平条形图。本节对基本水平条形图与层叠水平条形图的绘制进行讲解。图 2-21 所示为基本水平条形图，展示了万科 A、国农科技和世纪星源这 3 只股票在 2016 年净资产收益率的对比，数据来源是 Wind 资讯。本案例见文件 2.7_HBarChart_1.py。

本案例的代码如下。

```
# 2.7.1 基本案例
# Horizontal Bar Charts in Python
import plotly as py
import plotly.graph_objs as go
```

```
from plotly import tools
pyplt = py.offline.plot
data = [go.Bar(
          x=[29.41, 34.62, 30.16],
          y=['万科A', '国农科技', '世纪星源'],
          orientation = 'h'
)]
layout = go.Layout(
          title = '净资产收益率对比'
    )
figure = go.Figure(data = data, layout = layout)
pyplt(figure, filename='tmp/horizontal-bar.html')
```

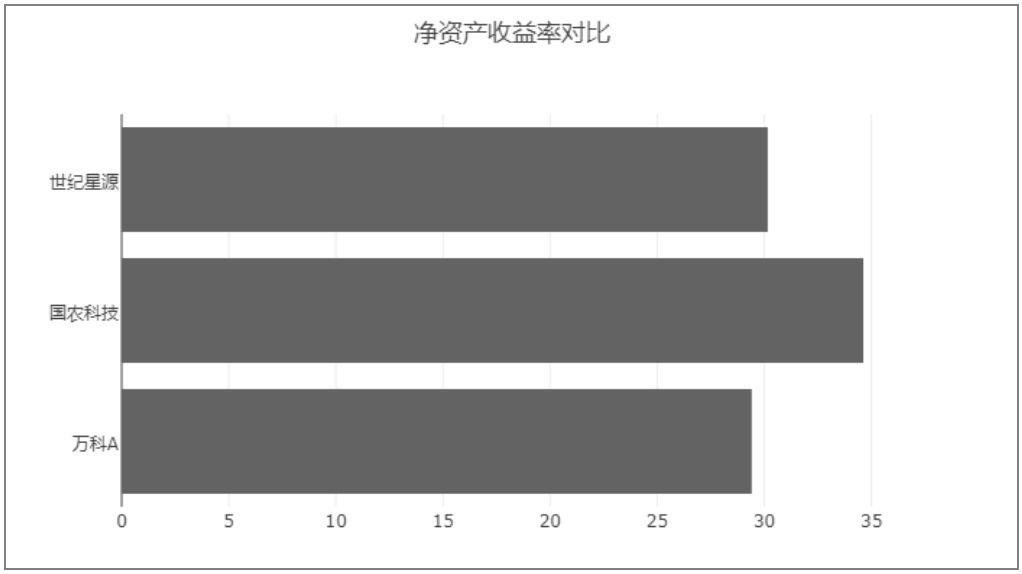

图 2-21　基本水平条形图

层叠水平条形图只需设置 barmode = 'stack'即可，相应数据和样式设置与基本水平条形图类似，层叠水平条形图的案例如图 2-22 所示，见文件 2.7_HBarChart_2.py。本案例展示了 AU.SHF、AG.SHF、CU.SHF 这 3 个期货品种对海通期货、永安期货及中信期货的多头持仓状况，数据来源是 Wind 资讯。

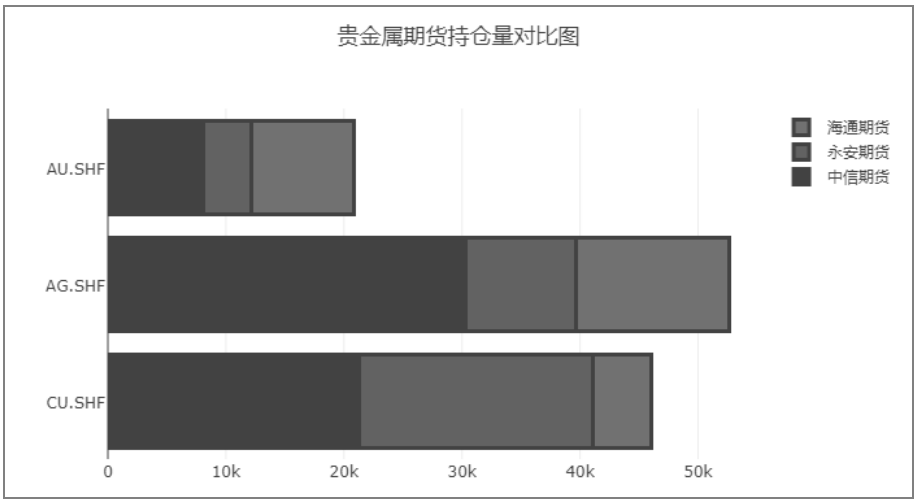

图 2-22　贵金属期货持仓量对比

本案例的代码如下。

```
# 2.7-2 应用案例
import plotly as py
import plotly.graph_objs as go
import numpy as np
import pandas as pd
from plotly import tools
# Colored Horizontal Bar Chart
pyplt = py.offline.plot
trace1 = go.Bar(
    y = ['CU.SHF', 'AG.SHF', 'AU.SHF'],
    x = [21258, 30279, 8056],
    name = '中信期货',
    orientation = 'h',
    marker = dict(
        color = '#104E8B',
        line = dict(
            color = '#104E8B',
            width = 3)
    )
)
trace2 = go.Bar(
```

```
        y = ['CU.SHF', 'AG.SHF', 'AU.SHF'],
        x = [19853, 9375, 4063],
        name = '永安期货',
        orientation = 'h',
        marker = dict(
            color = '#1874CD',
            line = dict(
                color = '#104E8B',
                width = 3)
        )
    )
    trace3 = go.Bar(
        y = ['CU.SHF', 'AG.SHF', 'AU.SHF'],
        x = [4959, 13018, 8731],
        name = '海通期货',
        orientation = 'h',
        marker = dict(
            color = '#1C86EE',
            line = dict(
                color = '#104E8B',
                width = 3)
        )
    )

    data = [trace1, trace2,trace3]
    layout = go.Layout(
        title = '贵金属期货持仓量对比图',
        barmode='stack'
    )

    fig = go.Figure(data=data, layout=layout)
    pyplt(fig, filename='tmp/marker-h-bar.html')
```

2.7.2 应用案例

本小节详细介绍 Plotly 官方网站提供的一个较复杂的应用案例，综合了水平条

形图与折线图的绘制、不同标签的添加、不同标注文字样式的设置、画布样式的设置等。为方便介绍，将逐段详细讲解代码功能，完整代码见文件 2.7_HBarChart_3.py，如图 2-23 所示为最终的绘图结果。

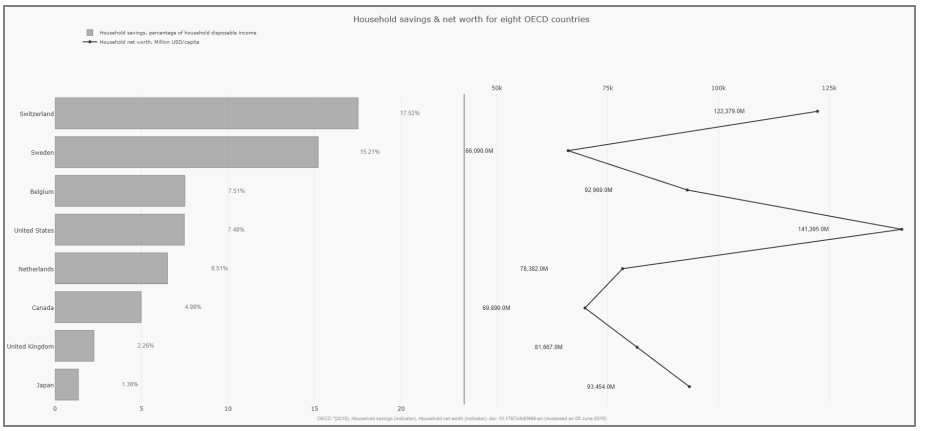

图 2-23　水平条形图应用案例

下面这部分代码的功能是加载工具包与数据，y_saving 中的数据用于左侧条形图，数据大小反映了条形的长度；y_net_worth 中的数据用于右侧折线图，每个数据对应折线中的数据点；x_saving 与 x_net_worth 中的数据对应左侧坐标轴上的标签。

```
    # 2.7-3 应用案例
    import plotly as py
    import plotly.graph_objs as go
    import numpy as np
    import pandas as pd
    from plotly import tools
    pyplt = py.offline.plot
    y_saving = [1.3586, 2.2623000000000002, 4.9821999999999997,
6.5096999999999996,
                7.4812000000000003, 7.5133000000000001, 15.2148,
17.520499999999998
                ] # 对应条形的长度
    y_net_worth = [93453.919999999998, 81666.570000000007,
69889.619999999995,
                   78381.529999999999, 141395.29999999999,
```

```
92969.020000000004,
                66090.179999999993, 122379.3] # 对应折线图上的值，从下到上
    x_saving = ['Japan', 'United Kingdom', 'Canada', 'Netherlands',
              'United States', 'Belgium', 'Sweden', 'Switzerland']
    x_net_worth = ['Japan', 'United Kingdom', 'Canada', 'Netherlands',
                'United States', 'Belgium', 'Sweden', 'Switzerland']
```

完成工具包与数据的加载后，在 trace0 中使用 Bar 函数对条形图进行数据与样式设置，在 trace1 中使用 Scatter 函数对折线图进行数据与样式设置。color 属性用于设置图形的颜色，已经在代码部分加以注释，其他参数可参考 2.6.7 节柱状图的参数，这里不再赘述。

```
    trace0 = go.Bar(
        x = y_saving,
        y = x_saving,
        marker = dict(
            color = 'rgba(50, 171, 96, 0.6)', # 条形颜色
            line = dict(
                color = 'rgba(50, 171, 96, 1.0)', # 条形边框颜色
                width = 1),
        ),
        name = 'Household savings, percentage of household disposable income',
        orientation = 'h',
    )
    trace1 = go.Scatter(
        x = y_net_worth,
        y = x_net_worth,
        mode = 'lines + markers',
        line = dict(
            color = 'rgb(128, 0, 128)'), # 折线颜色
        name = 'Household net worth, Million USD/capita',
    )
```

下面的代码分别对条形图与折线图的样式进行设置，yaxis1 与 xaxis1 分别对应左边条形图的 x 轴与 y 轴格式；showgrid 表示是否显示纵向或横向网格；showline 表示是否显示左侧或下方轴线；showticklabels 表示是否显示坐标轴上的标注；domain 限定了坐标轴范围；legend 用于设置图例，代码中包括了其位置与显示字体大小的

设置；margin 用于对整张图的留白区域进行设置，其中 l 表示 left，r 表示 right，t 表示 top，b 表示 bottom，分别控制四个方向；paper_bgcolor 用于设置整张图的背景颜色；plot_bgcolor 用于设置画布的背景颜色。

```
layout = dict(
    title = 'Household savings & net worth for eight OECD countries',
    # 左边图的 y 轴
    yaxis1 = dict(
        showgrid = False,          # 是否显示横向网格
        showline = False,          # 是否显示左侧轴线
        showticklabels = True,     # 是否显示坐标轴上的标注
        domain = [0, 0.85],
    ),
    # 右边图的 y 轴
    yaxis2 = dict(
        showgrid = False,
        showline = True,
        showticklabels = False,
        linecolor = 'rgba(102, 102, 102, 0.8)', # 左侧轴线颜色
        linewidth = 2,
        domain = [0, 0.85],
    ),
    # 左边图的 x 轴
    xaxis1 = dict(
        zeroline = False,     # 是否显示左侧轴线
        showline = False,     # 是否显示下方轴线
        showticklabels = True,
        showgrid = True,      # 是否显示纵向网格
        domain = [0, 0.42],
    ),
    # 右边图的 x 轴
    xaxis2 = dict(
        zeroline = False,
        showline = False,
        showticklabels = True,
        showgrid = True,
        domain = [0.47, 1],
```

```
            side = 'top', # 轴上标注在上方，默认为下方
            dtick = 25000, # 调整轴上标注的数值间隔，25000 表示相邻标注间隔数值为
25000
        ),
        legend = dict(
            x = 0.029,   # 图例 x 位置
            y = 1.038,   # 图例 y 位置
            font = dict(
                size = 10, # 图例字号大小
            ),
        ),
        margin = dict(
            l = 100, # 左侧空白大小
            r = 20,  # 右侧空白大小
            t = 70,  # 上方空白大小
            b = 70,  # 下方空白大小
        ),
        paper_bgcolor = 'rgb(248, 248, 255)', # 整张图片的背景颜色
        plot_bgcolor = 'rgb(248, 248, 255)',  # 绘图部分的背景颜色
    )
```

下面这部分代码对数据注释进行了设置，包括使用 font 设置标签字体、颜色与大小，使用 text 设置文字内容与样式，使用 showarrow 判断是否添加从标签到数据点的箭头等。

```
    annotations = []

    y_s = np.round(y_saving, decimals = 2) # 四舍五入至两位小数
    y_nw = np.rint(y_net_worth)           # 四舍五入至整数

    for ydn, yd, xd in zip(y_nw, y_s, x_saving): # 把数据对应起来
        # 为右侧折线图设置标签
        annotations.append(dict(xref = 'x2', yref = 'y2',
                                y = xd, x = ydn - 20000,
                                text='{:,}'.format(ydn) + 'M', # 从右向左，每隔三
位','
                                font = dict(family = 'Arial', size = 12,
                                        color = 'rgb(128, 0, 128)'), # 设置标签
```

```
字体、颜色与大小
                            showarrow = False)) # 是否添加从标签到数据点的箭头
        # 为左侧水平条形图设置标签
        annotations.append(dict(xref = 'x1', yref = 'y1',
                            y = xd, x = yd + 3,
                            text = str(yd) + '%',
                            font = dict(family = 'Arial', size = 12,
                                    color = 'rgb(50, 171, 96)'),
                            showarrow = False))
    # 下侧标签设置
    annotations.append(dict(xref = 'paper', yref = 'paper', # 设置文字样式
                        x = 0.3, y = -0.05, # 设置文字位置
                        text = 'OECD "' +
                             '(2015), Household savings (indicator), ' +
                             'Household net worth (indicator). doi: ' +
                             '10.1787/cfc6f499-en (Accessed on 05 June
2015)', # 设置图下方文字
                        font = dict(family = 'Arial', size = 10, # 设置图下
方文字格式
                                color = 'rgb(150,150,150)'),
                        showarrow = False))

    layout['annotations'] = annotations
```

下面这部分代码是对整体画布进行布局，用 rows = 1、cols = 2 划分为左右两块，shared_xaxes 与 shared_yaxes 用于设置左右两边的图是否共享同一个坐标轴。

```
    # Creating two subplots
    # shared_yaxes 不共享 y 轴，shared_xaxes 共享 x 轴，rows = 1、cols = 2 表示划
分为两块
    fig = tools.make_subplots(rows = 1, cols = 2,
                          shared_xaxes = True,
                          shared_yaxes = False)

    fig.append_trace(trace0, 1, 1)
    fig.append_trace(trace1, 1, 2)

    fig['layout'].update(layout)
    pyplt(fig, filename='tmp/oecd-networth-saving-bar-line.html')
```

2.7.3 参数解读

由于水平条形图是由柱状图修改一个参数 orientation = 'h'而来的，所以它们的其他参数是一样的，可以参考 2.6.7 节的相关内容。

2.8 甘特图

2.8.1 基本甘特图

甘特图（Gantt Chart）又称为横道图，通过条形来显示项目的进度、时间的安排等与时间相关的情况。本节讲解的甘特图案例来自对 Plotly 官方网站案例的修改，通过对代码与图的解读来了解与绘制甘特图。绘制甘特图的函数是 Plotly.figure_factory 中的 create_gantt，通过传递包含事件（Task）与开始、结束（Start、Finish）时间的数据来绘制图表。

下面的案例中包含三组数据，横坐标从左向右，时间不断延续，越接近于左侧的条形，其对应事件发生的时间越早；反之越晚。甘特图中条形的长度代表事件持续期的长短。本案例见文件 2.8_GanttChart_1.py，如图 2-24 所示为基本甘特图。

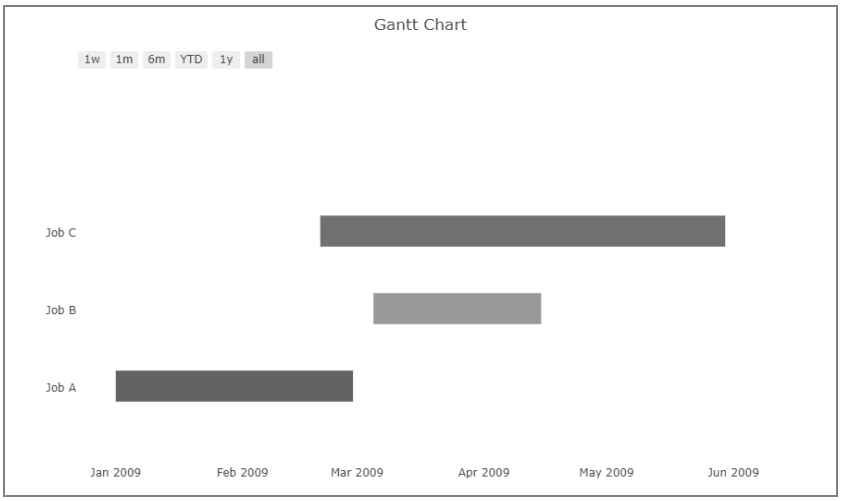

图 2-24　基本甘特图

本案例代码如下。

```
    # 2.8-1 基本案例
    import plotly as py
    import plotly.figure_factory as ff

    pyplt = py.offline.plot

    df = [dict(Task = "Job A", Start = '2009-01-01', Finish = '2009-02-28'),
          dict(Task = "Job B", Start = '2009-03-05', Finish = '2009-04-15'),
          dict(Task = "Job C", Start = '2009-02-20', Finish = '2009-05-30')]

    fig = ff.create_gantt(df,)
    pyplt(fig, filename='tmp/gantt-simple-gantt-chart.html')
```

2.8.2　甘特图（按数字索引）

将传入甘特图的数据按数字索引可以方便地对任务进行分类，本节案例的数据中包含了 6 项工作，每项工作后的 complete 属性即为其对应的索引值，在 create_gantt 函数中设置 index_col='Complete'，则有相同索引值的条形会呈现出相同的颜色，右侧的颜色条可以帮助用户根据颜色来判断任务大概对应的索引值是多少，这个数字的范围是 0～100，用来反映工作完成的进度，100 表示全部完成，0 表示没有进展。本案例见文件 2.8_GanttChart_2.py，如图 2-25 所示为按数字索引的甘特图。

本案例代码如下。

```
    # 2.8-2 应用案例
    import plotly as py
    import plotly.figure_factory as ff

    pyplt = py.offline.plot
    df = [dict(Task="Job A", Start='2009-01-01', Finish='2009-02-28',
Complete=10),
          dict(Task="Job B", Start='2008-12-05', Finish='2009-04-15',
Complete=10),
          dict(Task="Job C", Start='2009-02-20', Finish='2009-05-30',
```

```
Complete=50),
        dict(Task="Job D", Start='2009-03-20', Finish='2009-06-30',
Complete=50),
        dict(Task="Job E", Start='2009-01-12', Finish='2009-04-28',
Complete=100),
        dict(Task="Job F", Start='2009-03-07', Finish='2009-08-21',
Complete=100)]

    fig = ff.create_gantt(df,  index_col='Complete', show_colorbar=True)
    pyplt(fig, filename='tmp/gantt-numeric-variable.html')
```

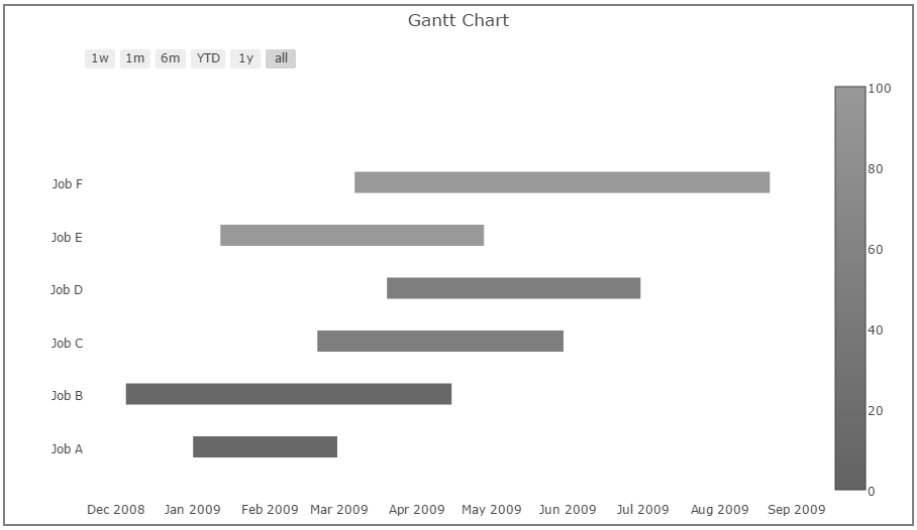

图 2-25　甘特图（按数字索引）

2.8.3　甘特图（按类别索引）

对甘特图的另一种索引方式是按照类别索引，本案例中包含 8 项工作，每项工作（数据）中的 Resource 代表此项工作所属的状态，状态分为 Complete、Incomplete 和 Not Started 三种，通过设置 create_gantt 函数中的 index_col='Resource' 即可完成通过 Resource 类别索引的目的。color 属性用于设置不同状态对应的颜色，注意需要以字典的格式传递给 create_gantt。本案例见文件 2.8_GanttChart_3.py，如图 2-26 所示为按类别索引的甘特图。

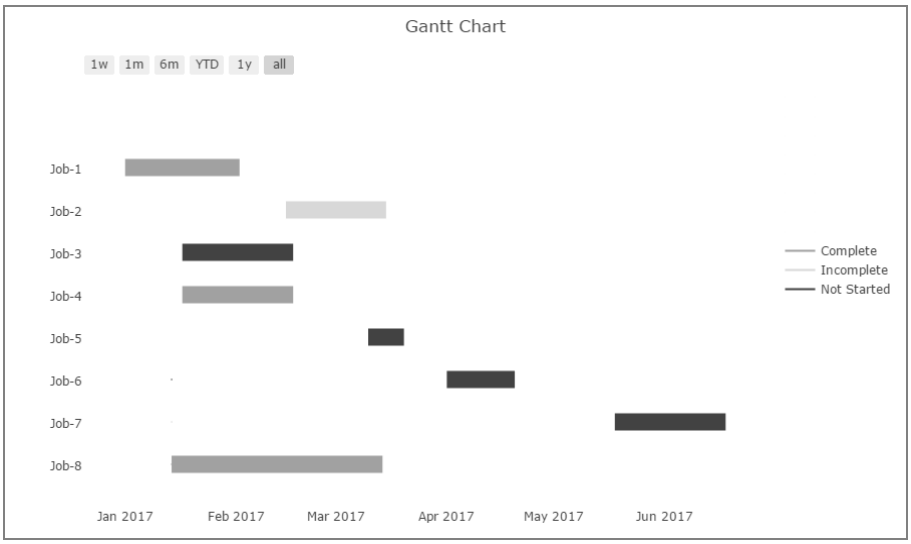

图 2-26　甘特图（按类别索引）

本案例代码如下。

```
    # 2.8-3 应用案例
    import plotly as py
    import plotly.figure_factory as ff
    pyplt = py.offline.plot

    df = [dict(Task="Job-1", Start='2017-01-01', Finish='2017-02-02',
Resource='Complete'),
          dict(Task="Job-2", Start='2017-02-15', Finish='2017-03-15',
Resource='Incomplete'),
          dict(Task="Job-3", Start='2017-01-17', Finish='2017-02-17',
Resource='Not Started'),
          dict(Task="Job-4", Start='2017-01-17', Finish='2017-02-17',
Resource='Complete'),
          dict(Task="Job-5", Start='2017-03-10', Finish='2017-03-20',
Resource='Not Started'),
          dict(Task="Job-6", Start='2017-04-01', Finish='2017-04-20',
Resource='Not Started'),
          dict(Task="Job-7", Start='2017-05-18', Finish='2017-06-18',
Resource='Not Started'),
          dict(Task="Job-8", Start='2017-01-14', Finish='2017-03-14',
```

```
Resource='Complete')]

    colors = {'Not Started': 'rgb(220, 0, 0)',
            'Incomplete': (1, 0.9, 0.16),
            'Complete': 'rgb(0, 255, 100)'}

    fig = ff.create_gantt(df, colors=colors, index_col='Resource',
group_tasks=True)
    pyplt(fig, filename='tmp/gantt-group-tasks-together.html')
```

2.8.4 应用案例

下面的案例是按类别索引的甘特图，注意此案例中任务起止时间的单位精确到了秒，并且对条形的样式进行了设置，使用 color 设置不同类别任务条的颜色，使用 bar_width 设置任务条的宽度，使用 showgrid_x 与 showgrid_y 设置是否显示横、纵坐标轴，以及使用 title 设置标题。本案例见文件 2.8_GanttChart_4.py，运行结果如图 2-27 所示。

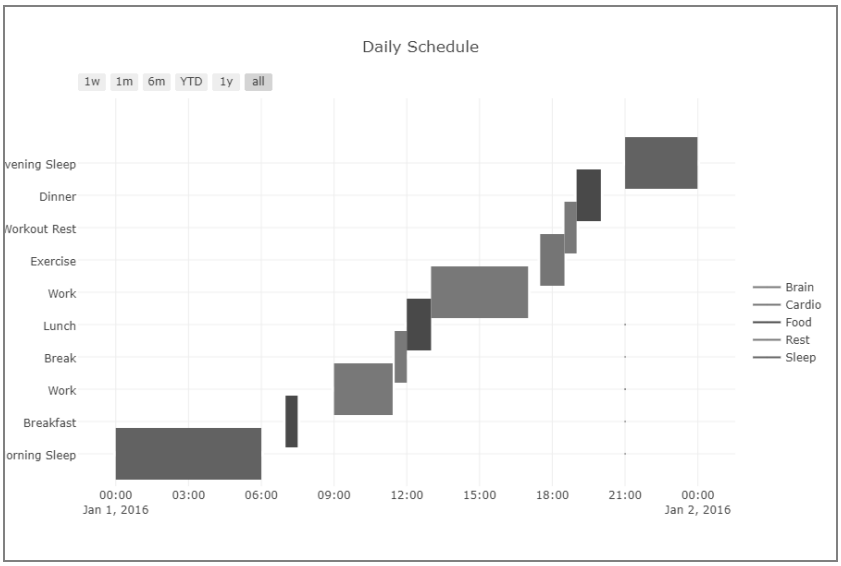

图 2-27　按类别索引的甘特图应用案例

本案例代码如下。

```
    # 2.8-4 应用案例
    import plotly as py
    import plotly.figure_factory as ff
    pyplt = py.offline.plot

    df = [
        dict(Task='Morning Sleep', Start='2016-01-01', Finish='2016-01-01
6:00:00', Resource='Sleep'),
        dict(Task='Breakfast', Start='2016-01-01 7:00:00',
Finish='2016-01-01 7:30:00', Resource='Food'),
        dict(Task='Work', Start='2016-01-01 9:00:00', Finish='2016-01-01
11:25:00', Resource='Brain'),
        dict(Task='Break', Start='2016-01-01 11:30:00', Finish='2016-01-01
12:00:00', Resource='Rest'),
        dict(Task='Lunch', Start='2016-01-01 12:00:00', Finish='2016-01-01
13:00:00', Resource='Food'),
        dict(Task='Work', Start='2016-01-01 13:00:00', Finish='2016-01-01
17:00:00', Resource='Brain'),
        dict(Task='Exercise', Start='2016-01-01 17:30:00',
Finish='2016-01-01 18:30:00', Resource='Cardio'),
        dict(Task='Post Workout Rest', Start='2016-01-01 18:30:00',
Finish='2016-01-01 19:00:00', Resource='Rest'),
        dict(Task='Dinner', Start='2016-01-01 19:00:00', Finish='2016-01-01
20:00:00', Resource='Food'),
        dict(Task='Evening Sleep', Start='2016-01-01 21:00:00',
Finish='2016-01-01 23:59:00', Resource='Sleep')
    ]

    colors = dict(Cardio = 'rgb(46, 137, 205)',
                  Food = 'rgb(114, 44, 121)',
                  Sleep = 'rgb(198, 47, 105)',
                  Brain = 'rgb(58, 149, 136)',
                  Rest = 'rgb(107, 127, 135)')

    fig = ff.create_gantt(df, colors = colors, index_col = 'Resource', title
```

```
= 'Daily Schedule',
                        bar_width = 0.8, showgrid_x = True, showgrid_y = True)
    pyplt(fig, filename='tmp/gantt-hours-minutes.html')
```

2.9 面积图

2.9.1 基本面积图

使用 Plotly 绘制面积图与绘制散点图和折线图相同，都使用 plotly graph_objs 中的 Scatter 函数，不同之处在于面积图对于 fill 属性的设置，相当于在折线图的基础上对图形进行填充。如图 2-28 所示为基本面积图，展示了两个不同交易策略的净值曲线，为了方便，策略的每日收益率非真实数据，而是根据 NumPy 随机生成的数字。在图 2-28 中，使用 mode 属性隐藏了面积图的边界线，见文件 2.9_AreaChart_1.py。

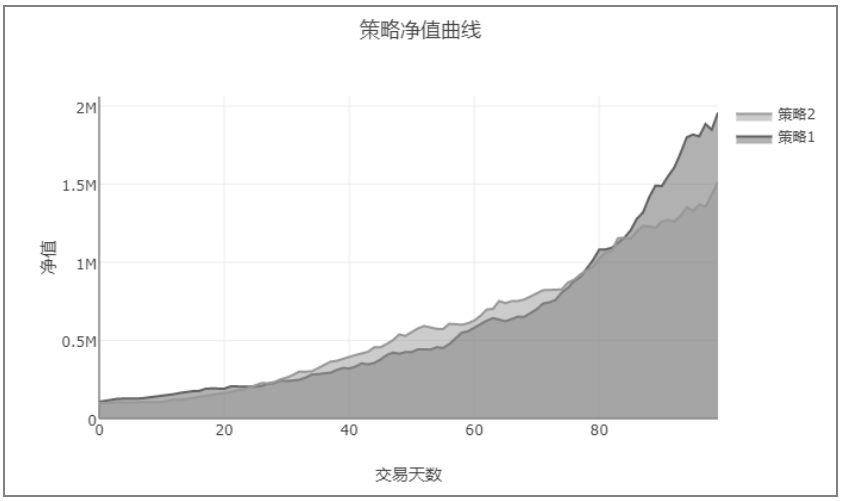

图 2-28 基本面积图

本案例的代码如下。

```
# 2.9-1 基本案例
import plotly as py
import plotly.graph_objs as go
```

```
import numpy as np
# 随机生成 100 个交易日的收益率
s1 = np.random.RandomState(8) # 定义局部种子
s2 = np.random.RandomState(9) # 定义局部种子
rd1 = s1.rand(100)/10 - 0.02
rd2 = s2.rand(100)/10 - 0.02

# 设定初始资金
initial1 = 100000
initial2 = 100000
total1 = []
total2 = []
for i in range(len(rd1)):
    initial1 = initial1*rd1[i] + initial1
    initial2 = initial2*rd2[i] + initial2
    total1.append(initial1)
    total2.append(initial2)

trace1 = go.Scatter(
    y = total1,
    fill = 'tonexty',
    mode= 'line', # none 无边界线
    name = "策略 1"
)
trace2 = go.Scatter(
    y = total2,
    fill = 'tozeroy',
    mode= 'line',# none 无边界线
    name = "策略 2"
)

data = [trace1, trace2]

layout = dict(title = '策略净值曲线',
              xaxis = dict(title = '交易天数'),
              yaxis = dict(title = '净值'),
              )
```

```
fig = dict(data = data, layout = layout)
pyplt(fig, filename='tmp/basic-area1.html')
```

2.9.2 内部填充面积图

内部填充面积图仅填充两条曲线交叉所形成的面积部分，同样通过设置 fill 属性来完成，只需在原面积图的基础上设置第一条曲线无填充效果，即 fill = None，再设置第二条曲线的填充的效果为 tonexty，即 fill = 'tonexty'。如图 2-29 所示为内部填充面积图，该图是在图 2-28 的基础上，通过修改 fill 属性完成的。本案例见文件 2.9_AreaChart_2.py。

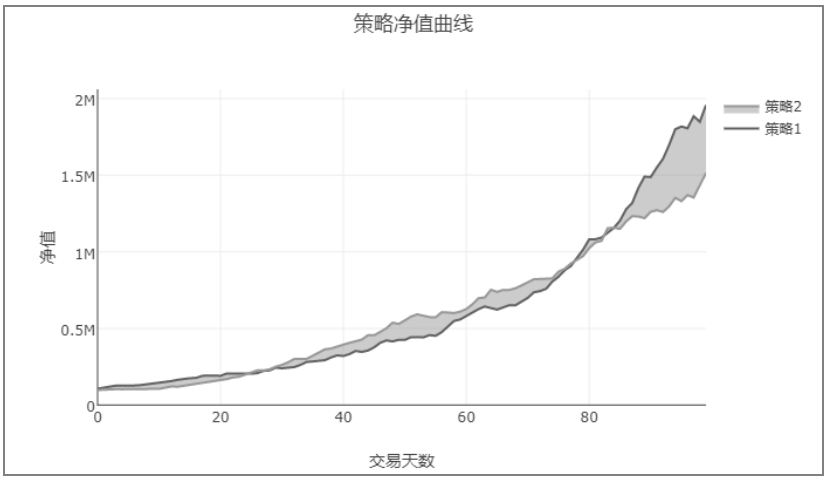

图 2-29　内部填充面积图

本案例的代码如下。

```
# 2.9-2 应用案例
# Interior Filling for Area Chart
import plotly as py
import plotly.graph_objs as go
import numpy as np
pyplt = py.offline.plot

# 随机生成 100 个交易日的收益率
```

```
s1 = np.random.RandomState(8) # 定义局部种子
s2 = np.random.RandomState(9) # 定义局部种子
rd1 = s1.rand(100)/10 - 0.02
rd2 = s2.rand(100)/10 - 0.02

# 设定初始资金
initial1 = 100000
initial2 = 100000
total1 = []
total2 = []
for i in range(len(rd1)):
    initial1 = initial1*rd1[i] + initial1
    initial2 = initial2*rd2[i] + initial2
    total1.append(initial1)
    total2.append(initial2)

trace1 = go.Scatter(
    y = total1,
    fill = None,
    mode= 'lines', # 无边界线
    name = "策略 1"
)
trace2 = go.Scatter(
    y = total2,
    fill = 'tonexty',
    mode= 'lines',# 无边界线
    name = "策略 2"
)

data = [trace1, trace2]

layout = dict(title = '策略净值曲线',
              xaxis = dict(title = '交易天数'),
              yaxis = dict(title = '净值'),
              )
fig = dict(data = data, layout = layout)
pyplt(fig, filename='tmp/basic-area2.html')
```

2.9.3 堆积面积图

堆积面积图与层叠柱状图类似，都展示了数据累加的效果，不同之处在于对数据的设置，下面讲解堆积面积图的绘制，以及与层叠柱状图绘制的不同之处。为了更好地展示可视化效果，在层叠柱状图的基础上对数据进行了调整，见文件 2.9_AreaChart_3.py。下面将堆积面积图中 data1 到 data4 中的数据单独列出，代码如下。

```
y = [32.52, 43.12, 43.47, 44.36, 33.11] # data1
y = [63.24, 54.33, 74.28, 63.91, 63.11] # data2
y = [83.24, 74.33, 93.91, 79.22, 83.11] # data3
y = [100, 100, 100, 100, 100] # data4
```

可以看出，data1 中的 y 表示基金 1 到基金 5 中股票投资占总体投资的比例，与层叠柱状图相同；data2 中的 y 表示基金 1 到基金 5 中股票投资与其他投资之和占总体投资的比例，而层叠柱状图 data2 中的 y 则表示基金 1 到基金 5 中其他投资占总体投资的比例。由此可见，从数据层面上讲，堆积面积图需要的是累加数据，而层叠柱状图只需要每部分的真实数据即可。

读者需要注意，在绘制层叠柱状图时需要设置 stack 模式，而绘制堆积面积图则不需要设置，这一点就是堆积面积图需要累加数据的原因，在本质上，堆积面积图的堆积效果是在同一个图形中绘制多个面积图来实现的。

堆积面积图的绘制效果如图 2-30 所示。

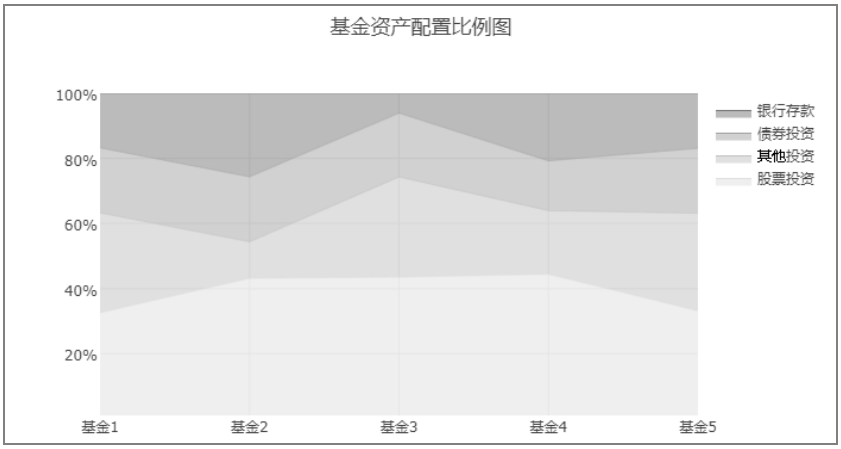

图 2-30　堆积面积图

本案例的代码如下。

```
# 2.9-3 应用案例
# Styled Pie Chart
import plotly as py
import plotly.graph_objs as go
# Data
data_1 = go.Scatter(
    x = ['基金 1', '基金 2', '基金 3', '基金 4','基金 5'],
    y = [32.52, 43.12, 43.47, 44.36, 33.11],
    name = '股票投资',
    mode = 'lines',
    line = dict(width=0.5,
            color = 'rgb(184, 247, 212)'),
    fill = 'tonexty'
)

data_2 = go.Scatter(
    x = ['基金 1', '基金 2', '基金 3', '基金 4','基金 5'],
    y = [63.24, 54.33, 74.28, 63.91, 63.11],
    name = '其他投资',
    mode = 'lines',
    line = dict(width=0.5,
            color = 'rgb(111, 231, 219)'),
    fill = 'tonexty'
)

data_3 = go.Scatter(
    x = ['基金 1', '基金 2', '基金 3', '基金 4','基金 5'],
    y = [83.24, 74.33, 93.91, 79.22, 83.11],
    name='债券投资',
    mode='lines',
    line=dict(width=0.5,
            color='rgb(127, 166, 238)'),
    fill='tonexty'
)

data_4 = go.Scatter(
    x = ['基金 1', '基金 2', '基金 3', '基金 4','基金 5'],
    y = [100, 100, 100, 100, 100],
    name='银行存款',
    mode='lines',
```

```
        line=dict(width=0.5,
                color='rgb(131, 90, 241)'),
        fill='tonexty'
    )

    data = [data_1, data_2, data_3, data_4]

    # Layout
    layout = go.Layout(
        title = '基金资产配置比例图',
        showlegend = True,
        yaxis = dict(
            range = [1, 100],
            dtick = 20,
            ticksuffix = '%'
        )
    )

    # Figure
    fig = go.Figure(data = data, layout = layout)
    pyplt(fig, filename = 'tmp/stacked-area-plot.html')
```

下面对代码中涉及的一些参数进行讲解。注意 data 中的 line 参数，这里的 line 用于设置折线的样式，只有在显示折线时才有设置的意义，即 mode = 'lines'时。本案例中还设置了宽度（width）与颜色（color），Layout 中的 yaxis 设置包含了 *y* 轴坐标的范围 range 是 0～100，坐标间隔 dtick 是 20，后缀符号 ticksuffix 是%。

至此，虽然已经可以对堆积面积图进行绘制，但是如果只知道原始数据，则需要计算出累积数据才能进行绘图，下面提供一个方法来完成累计数据的计算。

假设只知道 4 组原始数据 yi_org，通过使用句内 for 循环来计算累计数据，代码如下。

```
y0_org=[10, 0, 20, 10]
y1_org=[20, 30, 30, 10]
y2_org=[30, 30, 20, 20]
y3_org=[40, 40, 30, 60]

y0_stck=y0_org
y1_stck=[y0+y1 for y0, y1 in zip(y0_org, y1_org)]
```

```
    y2_stck=[y0+y1+y2 for y0, y1, y2 in zip(y0_org, y1_org, y2_org)]
    y3_stck=[y0+y1+y2+y3 for y0, y1, y2,y3 in zip(y0_org, y1_org,
y2_org,y3_org)]
```

2.10　直方图

2.10.1　基本直方图

使用 Plotly 绘制直方图需要用到 graph_objs 包中的 Histogram 函数。将数据赋值给函数中的 *x* 变量，即 *x* = data，即可绘制直方图；若将数据赋值给 *y* 变量，则绘制水平直方图。histnorm 是 Histogram 函数的另一个属性，默认状态下表示直方图的纵坐标落入区间内的样本数目；若设定 histnorm = 'probability'，则纵坐标变为落入区间内的样本频率。Histogram 函数的其他常用属性将在后面的案例中提到。下面的案例使用 NumPy 生成随机数，对基本的直方图进行实现，见文件 2.10_HistogramsChart_1.py。

基本直方图的绘制效果如图 2-31 所示。

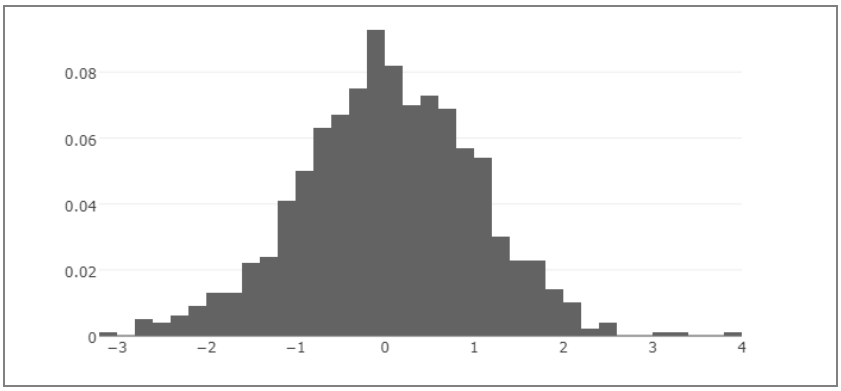

图 2-31　基本直方图

本案例的代码如下。

```
    # 2.10-1 基本案例
    # Basic Histogram
    import plotly as py
```

```
import plotly.graph_objs as go
import numpy as np

pyplt = py.offline.plot
s1 = np.random.RandomState(1)
x = s1.randn(1000)
data = [go.Histogram(x=x,
                     histnorm = 'probability')]
# y = x，水平直方图；histnorm='probability'，y 轴显示概率，没有则显示数目
pyplt(data, filename='tmp/basic_histogram.html')
```

2.10.2 重叠直方图

下面介绍重叠直方图的绘制，需要在 Layout 中设置 barmode 属性，将其改为 'overlay'，如果不对其进行设置，会出现 Plotly 默认将两个直方图的柱形宽度强制变窄的情况，以满足重叠部分显示的需求。与 Bar 函数等相同，Histogram 函数中也有 opacity 等通用属性，这里不再重复介绍。下面的案例使用 NumPy 生成分别服从正态分布（Randn）与卡方分布（Chisquare）的两组数据，并使用直方图的形式加以展现，见文件 2.10_HistogramsChart_2.py。

重叠直方图的绘制效果如图 2-32 所示。

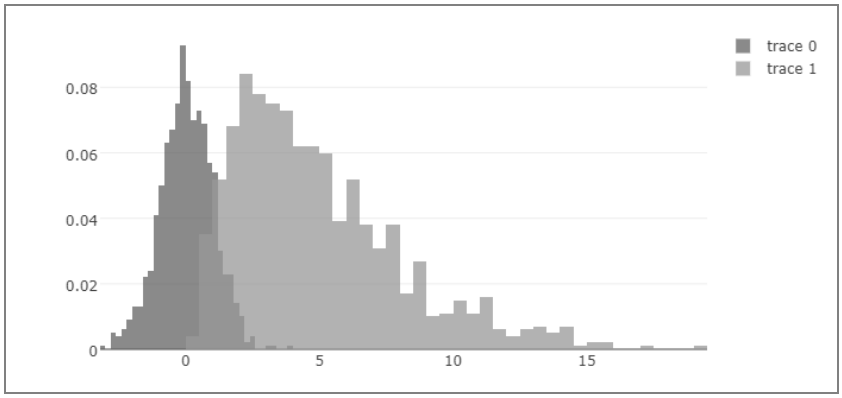

图 2-32　重叠直方图

本案例的代码如下。

```
# 2.10-2 应用案例
# Overlaid Histgram
import plotly as py
import plotly.graph_objs as go
import numpy as np

pyplt = py.offline.plot
s1 = np.random.RandomState(1)
x0 = s1.randn(1000)
x1 = s1.chisquare(5,1000)

trace1 = go.Histogram(
    x = x0,
    histnorm = 'probability',
    opacity = 0.75
)
trace2 = go.Histogram(
    x = x1,
    histnorm = 'probability',
    opacity = 0.75
)

data = [trace1, trace2]
layout = go.Layout(barmode='overlay')
fig = go.Figure(data = data, layout = layout)
pyplt(fig, filename='tmp/overlaid_histogram.html')
```

2.10.3　层叠直方图

绘制层叠直方图同样需要设置 barmode 属性，将其设置为'stack'。下面的案例使用 NumPy 生成两组相同的正态分布数据，trace0 显示的是第一组数据，trace1 显示的是第一组数据与第二组数据叠加的效果，见文件 2.10_HistogramsChart_3.py。

层叠直方图的绘制效果如图 2-33 所示。

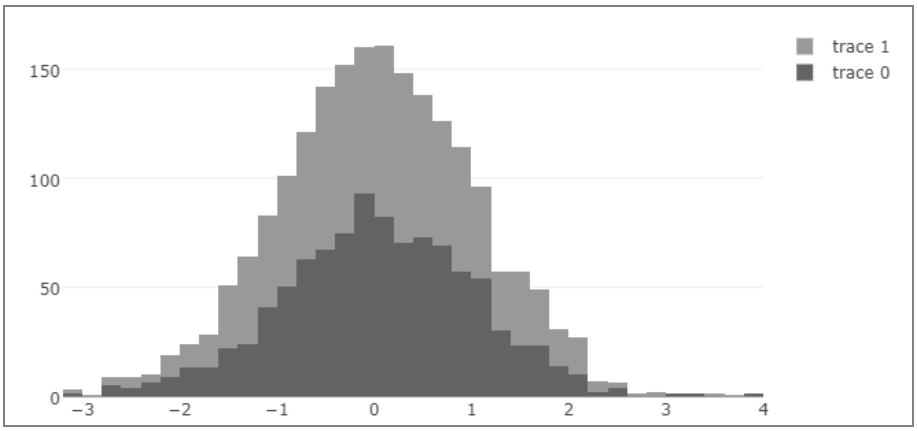

图 2-33　层叠直方图

本案例的代码如下。

```
# 2.10-3 应用案例
# Stacked Histograms
import plotly as py
import plotly.graph_objs as go
import numpy as np
pyplt = py.offline.plot

s1 = np.random.RandomState(1)
x0 = s1.randn(1000)
x1 = s1.randn(1000)

trace0 = go.Histogram(
    x=x0
)
trace1 = go.Histogram(
    x=x1
)
data = [trace0, trace1]
layout = go.Layout(barmode='stack')
fig = go.Figure(data=data, layout=layout)

pyplt(fig, filename='tmp/stacked_histogram.html')
```

2.10.4　累积直方图

累积直方图是直方图的累积形式，即第 *n*+1 个区间的展示数目是第 *n*-1 个区间的展示数目与第 *n* 个区间中实际样本数目之和，通过设置 Histogram 函数中的 cumulative 属性实现，即 cumulative=dict(enabled=True)。如图 2-34 所示是累积直方图的绘制效果，使用 NumPy 生成随机数据，见文件 2.10_HistogramsChart_4.py。

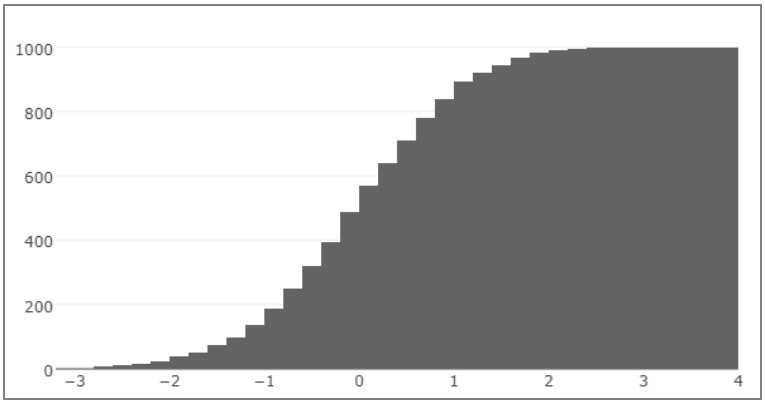

图 2-34　累积直方图

本案例的代码如下。

```
# 2.10-4 应用案例
# Cumulative Histogram
import plotly as py
import plotly.graph_objs as go
import numpy as np
pyplt = py.offline.plot

s1 = np.random.RandomState(1)
x1 = s1.randn(1000)
trace1 = [go.Histogram(
    x=x1,
    cumulative=dict(enabled=True)
)]
pyplt(trace1, filename='tmp/cumulative_histogram.html')
```

2.10.5 应用案例

plotly.figure_factory 包中的 distplot 函数将直方图与核函数估计可视化融合在一起，只需要将数据传递给 distplot 函数，即可完成相应的绘图。下面的案例使用 NumPy 生成分别服从柯西分布、泊松分布、Gamma 分布和指数分布的 4 组数据，并使用 distplot 函数展现直方图及对应的核密度估计函数，见文件 2.10_HistogramsChart_5.py。

直方图综合案例的绘制效果如图 2-35 所示。

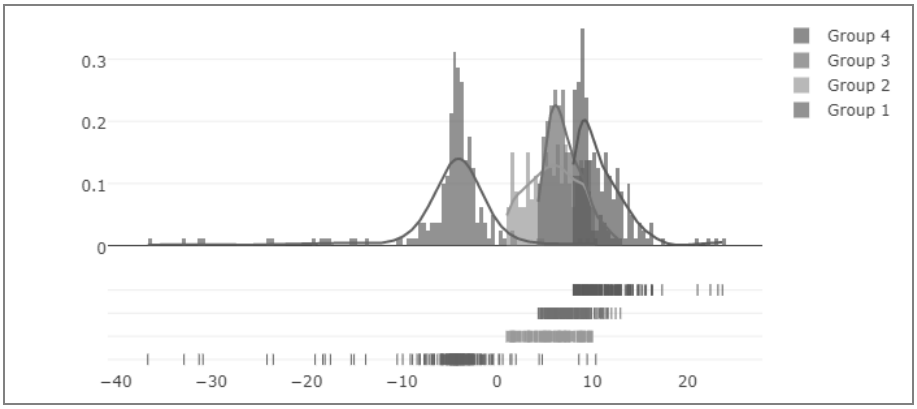

图 2-35　直方图综合案例

本案例的代码如下。

```
# 应用案例 2.10-5
# distplot
import plotly as py
import plotly.graph_objs as go
import numpy as np
import plotly.figure_factory as ff
pyplt = py.offline.plot

# Add histogram data
s1 = np.random.RandomState(12)
# 柯西分布
x1 = s1.standard_cauchy(200) - 4
# 泊松分布
```

```
x2 = s1.uniform(1,10,200)
# Gamma 分布
x3 = s1.standard_gamma(3,200) + 4
# 指数分布
x4 = s1.exponential(3,200) + 8

# Group data together
hist_data = [x1, x2, x3, x4]

group_labels = ['Group 1', 'Group 2', 'Group 3', 'Group 4']

# Create distplot with custom bin_size
fig = ff.create_distplot(hist_data, group_labels, bin_size=.4)

# Plot!
pyplt(fig, filename='tmp/Distplot_with_Multiple_Datasets.html')
```

2.10.6　参数解读

下面讲解直方图综合案例中使用的 create_distplot 函数和 histogram 函数的常用参数。

- histnorm：设置纵坐标显示格式，可选参数有""、percent、probability、density、probability density。若为 percent 或 probability，则表示纵坐标显示样本占总体的百分比，所有矩形的高相加为 100%；若为 density，则每个小矩形的面积为所在范围内的样本的数量，所有矩形面积相加为样本总数；若为 probability density，则每个小矩形的面积为所在范围内的样本占总体的比例，所有面积数值相加为 1。
- histfunc：指定分组函数，可选参数有 count、sum、avg、min、max。若为 count，则按照统计样本落在区间内的个数操作；若为 sum、avg、min、max，则依次对样本区间内的样本求和、取平均值、求最小值和求最大值。
- orientation：设置图形的方向，有 v 和 h 两个可选参数，v 表示垂直显示，h 表示水平显示。
- cumulative：累积直方图参数，有 enabled、directio 和 currentbin 三个关键字。其中，enabled 是布尔型，设置为 True 会显示累积直方图，设置为 False 则不对频

率或频数进行累积；direction 用于设置累积方向，确定频率是从 1～0（降序），还是从 0 到 1（升序）；currentbin 有三个选择，即 include、exclude、half，为了防止偏差，一般选择 half。

- autobinx：布尔型，是否自动划分区间。
- nbinsx：整型，最大显示区间数目。
- xbins：设置划分区间属性，有 start、end、size 三个关键字。start 设置起始坐标，end 设置终止坐标，size 设置区间长度。

2.11 饼图

2.11.1 基本饼图

使用 Plotly 绘制饼图需要使用 graph_objs 中的 Pie 函数。Pie 函数中最常用的两个属性一个是 values，用于赋给其需要可视化的数据；另一个是 labels，表示不同数据对应的标签。样式设置与前面几种图形的样式设置类似，这里不再赘述。如图 2-36 所示是饼图的绘制效果，展示了基金的资产配置结构，见文件 2.11_PieChart_1.py。

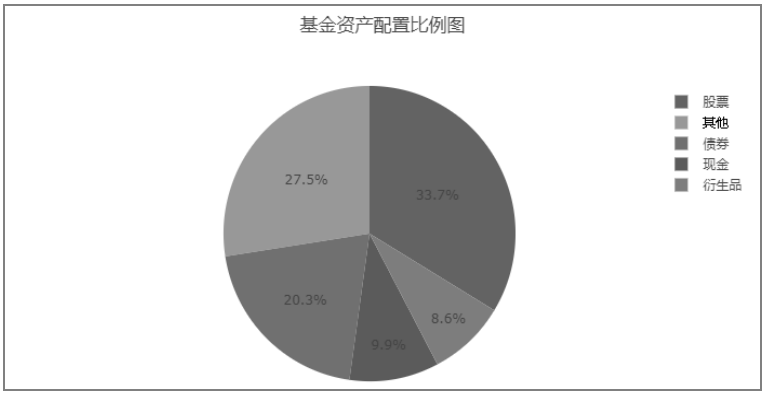

图 2-36 饼图

本案例的代码如下。

```
# 2.11-1 Basic Pie Chart
import plotly as py
```

```
import plotly.graph_objs as go
pyplt = py.offline.plot
labels = ['股票','债券','现金','衍生品','其他']
values = [33.7,20.33,9.9,8.6,27.47]
trace = [go.Pie(labels=labels, values=values)]
layout = go.Layout(
    title = '基金资产配置比例图',
)
fig = go.Figure(data = trace, layout = layout)
pyplt(fig, filename='tmp/basic_pie_chart.html')
```

2.11.2　环形饼图

绘制环形饼图，只需在 Pie 函数中设置控制环形中心空白大小的 hole 属性即可实现。Pie 函数中的 hoverinfo 属性用于控制当用户将鼠标指针放到环形图上时显示的内容，设置为“label + percent”表示显示标签加数据所占的比例。如图 2-37 所示是环形饼图最简单的实现，展示了工作进度，见文件 2.11_PieChart_2.py。

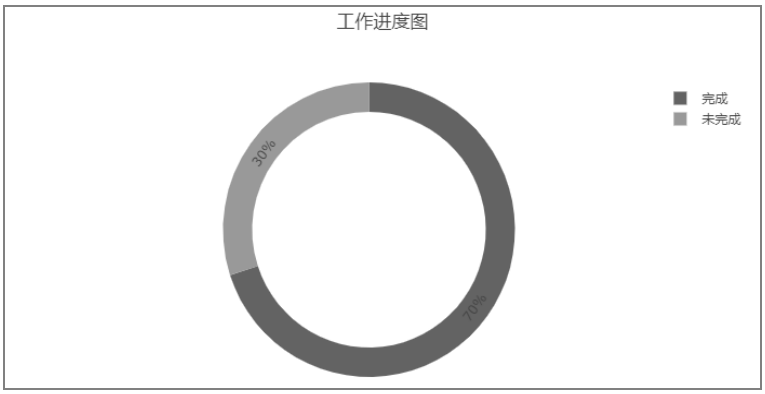

图 2-37　环形饼图

本案例的代码如下。

```
# 2.11.2 Basic Pie Chart
import plotly as py
import plotly.graph_objs as go
pyplt = py.offline.plot
```

```
labels = ['完成','未完成']
values = [0.7,0.3]
trace = [go.Pie(
    labels = labels,
    values = values,
    hole =  0.7,
    hoverinfo = "label + percent")]
layout = go.Layout(
    title = '工作进度图'
)
fig = go.Figure(data = trace, layout = layout)
pyplt(fig, filename='tmp/basic_pie_chart.html')
```

2.11.3 样式设置

对饼图颜色与样式的设置通过下面这个案例来说明。如图 2-38 所示的饼图展示了浦发银行五大流通股东的持股比例，本案例包括设置饼图的颜色、边框和标签等属性，见文件 2.11_PieChart_3.py。

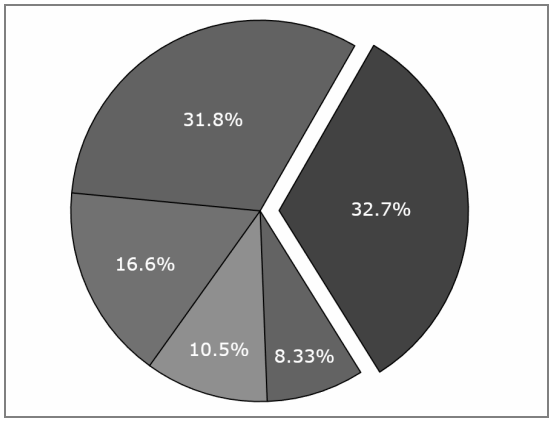

图 2-38　浦发银行五大流通股东持股比例图

本案例的代码如下。

```
# 2.11-3 Styled Pie Chart
import plotly as py
import plotly.graph_objs as go
```

```
pyplt = py.offline.plot
labels = ['上海国际集团有限公司', '中国移动通信集团',\
 '富德生命人寿-传统', '富德生命人寿-资本金', '上海上国投资产管理有限公司']
values = [4222533311, 4103763711, 2138028672, 1356332558, 1073516173]
colors = ['#104E8B', '#1874CD', '#1C86EE', '#6495ED']

trace = [go.Pie(labels = labels,
              values = values,
              rotation = 30,
              opacity = 1,
              showlegend = False,
              pull = [0.1,0,0,0,0],
              hoverinfo = 'label+percent',
              textinfo = 'percent', # textinfo = 'value',
              textfont = dict(size = 30, color = 'white'),
              marker = dict(colors = colors,
                  line = dict(color = '#000000', width = 2)))]
fig = go.Figure(data = trace)
pyplt(trace, filename='tmp/styled_pie_chart.html')
```

下面对 Pie 函数中涉及的参数进行讲解。通过设置 rotation 参数可以对饼图进行旋转，其取值为 0～360；opacity 参数用于设置透明度；showlegend 参数是布尔类型，True 表示展示图例，False 表示隐藏图例；pull 参数用于设置组成饼图的各个扇形的突出程度，0 表示不突出，本例中通过设置 pull 参数为[0,1,0,0,0,0]，使其中一个扇形突出；textinfo 参数用于设置显示在扇形上的是具体数值（value）还是比例（percent）；textfont 参数用于设置所显示内容的样式，本例中设置了字体大小（size）和字体颜色（color）；marker 参数用于设置每个扇形的样式，其中 colors 用于设置颜色，line 用于设置扇形边框的样式。

2.11.4　应用案例

本节对饼图进行实战应用。如图 2-39 所示，包含 4 个饼图，展示了有色金属板块 AU、AG 两大合约的多空持仓状况。本案例包括绘制子图、添加标签、调整位置等，见文件 2.11_PieChart_4.py。

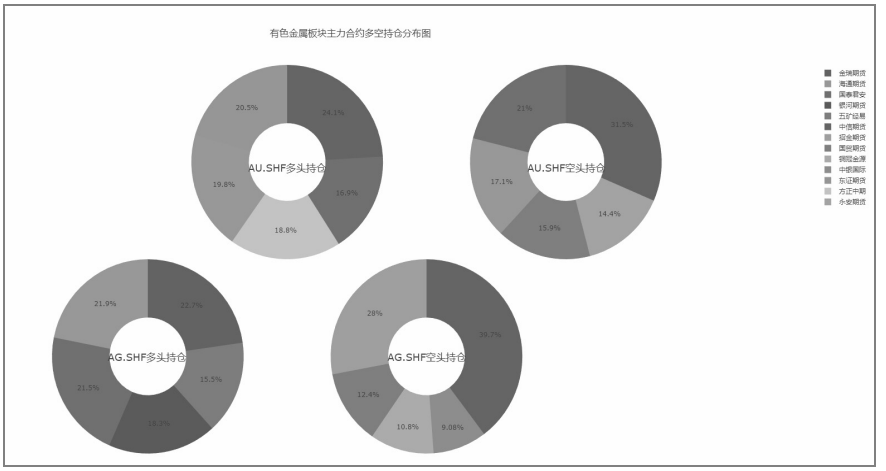

图 2-39　饼图应用案例

本案例的代码如下。

```
# 2.11-4 应用案例
import plotly as py
import plotly.graph_objs as go
# Donut Chart
pyplt = py.offline.plot
fig = {
  "data": [
    {
      "values": [9884, 9510, 9363, 7961, 6755],
      "labels": [
          '金瑞期货',
          '海通期货',
          '国泰君安',
          '银河期货',
          '五矿经易'
      ],
      'domain': {'x': [0, .6],
                 'y': [0, .5]},
      "name": "AU.SHF 多头持仓",
      "hoverinfo":"label + percent + name",
      "hole": .4,
      "type": "pie"
    },
```

```
{
  "values": [8847, 6229, 2764, 2406, 2022],
  "labels": [
     '中信期货',
     '招金期货',
     '国贸期货',
     '铜冠金源',
     '中银国际'
  ],
  'domain': {'x': [.2, 1],
          'y': [0, .5]},
  "name": "AU.SHF 空头持仓",
  "hoverinfo":"label + percent + name",
  "hole": .4,
  "type": "pie"
},
{
  "values": [14393, 12220, 11824, 11233, 10072],
  "labels": [
     '中信期货',
     '东证期货',
     '海通期货',
     '方正中期',
     '国泰君安'
  ],
  'domain': {'x': [0, .9],
          'y': [.5, 1]},
  "name": "AG.SHF 多头持仓",
  "hoverinfo":"label + percent + name",
  "hole": .4,
  "type": "pie"
},
{
  "values": [30983, 20699, 16781, 15686, 14198],
  "labels": [
     '中信期货',
     '国泰君安',
```

```
            '海通期货',
            '国贸期货',
            '永安期货'
        ],
        'domain': {'x': [0.5, 1],
                'y': [.5, 1]},
        "name": "AG.SHF 空头持仓",
        "hoverinfo":"label + percent + name",
        "hole": .4,
        "type": "pie"
      }],

    "layout": {
         "title":"有色金属板块主力合约多空持仓分布图",
         "annotations": [
             {
                 "font": {
                     "size": 18
                 },
                 "showarrow": False,
                 "text": "AU.SHF 多头持仓",
                 "x": 0.45,
                 "y": 0.754
             },
             {
                 "font": {
                     "size": 18
                 },
                 "showarrow": False,
                 "text": "AU.SHF 空头持仓",
                 "x": 0.794,
                 "y": 0.754
             },
             {
                 "font": {
                     "size": 18
                 },
```

```
                "showarrow": False,
                "text": "AG.SHF 多头持仓",
                "x": 0.255,
                "y": 0.23
            },
            {
                "font": {
                    "size": 18
                },
                "showarrow": False,
                "text": "AG.SHF 空头持仓",
                "x": 0.6,
                "y": 0.23
            }
        ]
    }
}
pyplt(fig, filename='tmp/donut.html')
```

data 部分有四个大括号（{}），每个大括号代表一个子饼图，例如下面的代码就表示一个子饼图。

```
{
      "values": [9884, 9510, 9363, 7961, 6755],
      "labels": [
          '金瑞期货',
          '海通期货',
          '国泰君安',
          '银河期货',
          '五矿经易'
      ],
      'domain': {'x': [0, .6],
                 'y': [0, .5]},
      "name": "AU.SHF 多头持仓",
      "hoverinfo":"label + percent + name",
      "hole": .4,
      "type": "pie"
}
```

其中，values 中是饼图的数据；labels 包含饼图中每一块扇形的标签；domain 表示饼图的位置；name 是子饼图的名称；hoverinfo 用于设置将鼠标指针移至扇形时显示的内容，包括每个扇形的标签、所占比例及所属子饼图的名称；hole 用于设置圆环的大小；type 用于声明是饼图。

除此之外，Layout 部分的 annotations 属性同样也包含了四个大括号（{}），每个大括号代表一个子饼图中心的文字设置，如下所示。

```
{
     "font": {
     "size": 18
     },
    "showarrow": False,
    "text": "AU.SHF 多头持仓",
    "x": 0.45,
    "y": 0.754
}
```

其中，font 中的 size 属性用于设置文本的大小，showarrow 用于设置是否显示箭头，text 中可以输入展示的文本内容，x 与 y 用于设置文本所在的位置。

2.11.5 参数解读

在 Plotly 模块库中使用 Pie 函数绘制饼图，常用属性如下。

- direction：饼图方向，有 clockwise（顺时针）和 counterclockwise（逆时针），默认值为 counterclockwise。
- dlabel：饼图图标步进值，默认值为 1。
- domain：范围，设置各个扇区的大小。
- hole：设置环形饼图内径孔的半径，范围是 0～1，默认值为 0，参数是与外径的比值。
- label0：生成一组扇区图标的起点数字，默认值为 0。
- labels：每个扇区的图标字符串数组列表，一般用 Pandas 的时间索引值。
- legendgroup：图标参数，默认是空字符串。
- marker：数据节点参数，包括大小、颜色、格式等。

- name：名称参数。
- opacity：透明度参数，范围是 0～1。
- pull：凸出扇区的比例，范围是 0～1，默认值为 0。
- pullsrc：各个扇区比例数组列表。
- rotation：扇区旋转角度，范围是 0～360，默认值为 0（12 点位置）。
- showlegend：布尔变量，用于切换图标显示。
- sort：布尔变量，扇区排序开关。
- values：每个扇区的数值大小。

2.12　更多案例

前面详细介绍了 Plotly 基本的绘图方法，这些绘图方法基本可以满足用户在可视化应用中的大部分绘图需求。当然，Plotly 中不止这些绘图方法，本章仅介绍一些最常用的。如果这些基本方法不能满足用户的需求，则可以登录官方网站（https://plot.ly/python/）查找，一般情况下都可以解决问题。

如图 2-40 所示是 Plotly 官方网站中的案例。

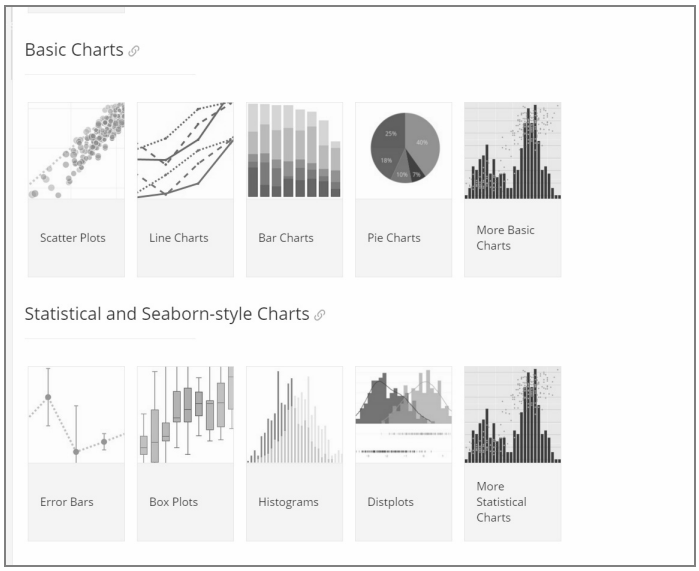

图 2-40　Plotly 官方网站中的案例

2.13 Plotly对象概览

经过前面的学习，读者对 Plotly 的基本绘图方法有了一些了解。本节介绍 Plotly 的对象（Objects），Plotly 的对象除各种类型的图表外，还包括数据库连接和数据分析工具。

大体而言，Plotly 的绘图对象包括以下类型。

- 2D 平面图表。
- 3D 立体图表。
- Maps，地图类型的图表。
- WebGL 格式图表。
- Layout，画面布局函数。
- Axis，坐标工具，包括 2D/3D 平面坐标、立体坐标和极坐标（AngularAxis）。
- Annotation，图表注释工具。
- Error，误差调整工具，包括 x、y、z 三种坐标的误差修正。
- Figure，图形整合工具。
- Font，字体工具。
- Legend，图例工具。
- Margin，图表边缘间距工具。
- Marker，标记工具。
- 金融行业通用的 OHLC 数据格式工具。
- Pointcloud，点云工具。
- RadialAxis，纵横比调整工具。
- Array，数据坐标列表。
- Data，图表数据整合工具。
- Stream，实时图表数据流工具。
- Scene，场景工具。

1. 2D 平面绘图对象

Plotly 的 2D 平面绘图对象如下。

- Angularaxis，极坐标图表。
- Area，区域图。

- Bar，条形图。
- Box，盒形图，又称箱线图、盒子图、箱图。
- Candlestick 与 OHLC，金融股票行业常用的 K 线图与 OHLC 曲线图。
- ColorBar，彩条图。
- Contour，轮廓图（等高线图）。
- Choropleth，等值线图。
- Line，曲线图。
- Heatmap，热点图。
- Histogram，直方图。
- Histogram 2d，2D 平面直方图。
- Histogram 2d Contour，二维轮廓直方图。
- Pie，饼图。
- Scatter，坐标分布图（包括线形图、散点图）。

2．3D 立体绘图对象

Plotly 的 3D 立体绘图对象如下。

- Scatter3D，3D 立体散点图（包括线形图）。
- Surface，表面图。
- Mesh3D，3D 立体网格图。
- Pointcloud，点云图。

3．Maps 地图

- ScatterGeo，基于 GEO 地图模式的散点图（包括线形图）。
- Choropleth，立体等值线图。
- Scattermapbox，基于地图的散点图。

4．WebGL 格式图形

- ScatterGL，WebGL 格式散点图（包括线形图）。

5．图表辅助工具

Plotly 绘图模块库的辅助函数如下。

- Annotation，图表注释工具。

- Axis，坐标工具，包括 2D/3D 平面坐标、立体坐标和极坐标（AngularAxis）。
- Annotation，图表注释工具。
- Error，误差调整工具，包括 *x*、*y*、*z* 三种坐标的误差修正。
- Figure，图形整合工具。
- Font，字体工具。
- Legend，图例工具。
- Margin，图表边缘间距工具。
- Marker，标记工具。
- RadialAxis，纵横比调整工具。
- Array，数据坐标列表。
- Scatterternary，散点三元图。
- Data，图表数据整合工具。
- Stream，实时图表数据流工具。
- Scene，场景工具。
- XBins、YBins：*x* 轴、*y* 轴辅助参数。
- Layout，画面布局函数。

6．Layout 画面布局函数

Layout 画面布局函数功能强大，看起来很复杂，其实只要掌握了相关的参数，设置很简单，都是字典模式，如图 2-41 所示是 Layout 画面布局函数官方文档的相关截图说明。

图 2-41　Layout 画面布局函数官方文档的相关截图说明

由图 2-41 可以看出，Layout 画面布局函数的有关参数如下。

- 3D Axes，三维坐标（*x* 轴、*y* 轴、*z* 轴）。
- Axes，平面坐标（*x* 轴、*y* 轴）。
- File Settings，文件设置。
- Horizontal Legends，水平标记。
- Images，插入图片，一般作为背景或地图数据。
- Inset Plots，插入图形。
- LaTeX，LaTeX 专业排版数据。
- Legends，图例设置。
- Logos，绘制格式标志。
- Multiple Axes，多轴设置。
- Setting Graph Size，设置图形的大小。
- Setting the Title, Legend Entries, and Axis Titles，设置标题、图例条目和轴标题。
- Subplots，子图设置。
- Text and Annotations，文本和注释。
- Text and Font Styling，文本和字体样式。

读者可以使用 help 命令查看 Layout 函数的参数与属性，代码如下。

```
import plotly.graph_objs as pygo

print(help( pygo.Layout))
```

输入以上代码时，注意大小写，Layout 画面布局函数的首字母是大写。

help 函数输出的信息很多，其中开头部分的代码如下。

```
    |  Valid attributes for 'layout' at path [] under parents ():
    |
    |      ['direction', 'yaxis', 'angularaxis', 'hiddenlabels', 'xaxis',
'images',
    |      'smith', 'mapbox', 'radialaxis', 'bargroupgap', 'sliders',
'titlefont',
    |      'geo', 'ternary', 'scene', 'dragmode', 'barmode', 'annotations',
    |      'calendar', 'hidesources', 'boxgroupgap', 'barnorm', 'legend',
    |      'boxmode', 'width', 'autosize', 'title', 'paper_bgcolor',
'margin',
```

```
    |     'hiddenlabelssrc', 'orientation', 'plot_bgcolor', 'height',
    |     'showlegend', 'font', 'shapes', 'separators', 'boxgap',
'hovermode',
    |     'bargap', 'updatemenus']
```

其中，两个中括号之间的部分就是 Layout 画面布局函数的属性和参数。

Matplotlib 绘图模块库和其他 Python 模块库的内置函数，也可以使用这种方法查看有关函数的内置属性。

下面根据以上输出信息，简单介绍 Layout 画面布局函数的属性和参数。

- angularaxis，角轴。
- annotations，注释。
- autosize，自动调整大小。
- bargap，（条形图）柱间距。
- bargroupgap，（条形图）柱组间距。
- barmode，条形图模式。
- barnorm，条形图参数。
- boxgap，盒子图间距。
- boxgroupgap，盒子组间距。
- boxmode，箱型图模式。
- calendar，日历。
- direction，方向。
- dragmode，图形拖动模式。
- font，字体。
- geo，地理参数。
- height，高度。
- hiddenlabels，隐藏图标。
- hiddenlabelssrc，隐藏图标参数数组列表。
- hidesources，隐藏数据源。
- hovermode，鼠标指针悬停模式。
- images，图像。
- legend，图标。
- mapbox，地图模式。

- margin，图表边缘间距。
- orientation，方向。
- paper_bgcolor，地图背景颜色。
- plot_bgcolor，图形背景颜色。
- radialaxis，纵横比。
- scene，场景。
- separators，分离参数。
- shapes，形状。
- showlegend，图例显示。
- sliders，滑块。
- smith，smith 参数。
- ternary，三元参数。
- title，标题。
- titlefont，标题字体。
- updatemenus，菜单更新。
- width，宽度。
- xaxis，*x* 轴。
- yaxis，*y* 轴。

Layout 画面布局函数主要用于设置图形外观，比如标题、横纵坐标轴、图例、图形外边距等属性，这些属性包括字体、颜色、尺寸等，更多 Layout 画面布局函数的有关资料，请读者浏览 Plotly 网站的在线文档，网址是 https://plot.ly/python/#layout-options。

7．Annotation 注释函数

Annotation 注释函数用于为图形整体和数据节点添加注解文本，主要属性和参数如下。

- align，对齐方式。
- arrowcolor，箭头颜色。
- arrowhead，箭头模式，共有 9 种模式。
- arrowsize，箭头大小。
- arrowwidth，箭头宽度。

- ax，*x* 轴坐标参数。
- axref，*x* 轴坐标辅助参数。
- ay，*y* 轴坐标参数。
- ayref，*y* 轴坐标辅助参数。
- bgcolor，背景颜色。
- bordercolor，边框颜色。
- borderpad，边框排列方式。
- borderwidth，边框宽度。
- font，字体。
- opacity，透明度。
- ref，辅助参数。
- showarrow，显示箭头开关。
- text，文本。
- textangle，文本角度。
- visible，注解显示开关。
- x、y：*x* 轴坐标、*y* 轴坐标。
- xanchor、yanchor：*x* 轴坐标锚点、*y* 轴坐标锚点。
- xref、yref：*x* 轴坐标参考参数、*y* 轴坐标参考参数；如果参考参数等于“paper”（画纸模式），则注释文字的 *x*、*y* 坐标分别以画纸左下角（0,0）为原点、以右上角（1,1）为坐标最大值，*x*、*y* 采用小数形式；如果参考参数等于“x”、“y”，则表示使用 *x*、*y* 数据数组的坐标参数作为注释字符串的坐标。

8．Axis 坐标轴参数设置

在绘制图形时，经常会根据实际情况调整坐标轴参数，Plotly 绘图模块库中设置坐标的参数有：xaxis（*x* 轴参数）、yaxis（*y* 轴参数）、zaxis（*z* 轴参数）。

有关坐标轴参数的属性说明如下。

- anchor，锚点。
- autorange，自动范围。
- autotick，自动刻度。
- backgroundcolor，背景颜色。
- calendar，日历模式。

- categoryarray，分类数据参数。
- categoryarraysrc，分类数据参数源。
- categoryorder，分类数据模式。
- color，颜色。
- domain，范围。
- dtick，分类数据坐标刻度步进值。
- exponentformat，指数格式。
- fixedrange，固定范围。
- gridcolor，网格线颜色。
- gridwidth，网格线宽度。
- hoverformat，鼠标指针悬停格式。
- linecolor，线条颜色。
- linewidth，线宽。
- mirror，镜像模式，包括 true、ticks、false、all、allticks 等属性。
- nticks，刻度之间的间隔数。
- overlaying，图层重叠。
- position，位置。
- range，范围。
- rangemode，范围模式。
- rangeselector，范围选择。
- rangeslider，范围滑块。
- separatethousands，千分位分隔。
- showaxeslabels，显示 *x* 轴图标开关。
- showbackground，显示背景开关。
- showexponent，显示指数开关。
- showgrid，显示网格开关。
- showline，显示线条开关。
- showspikes，显示异常开关。
- showticklabels，显示刻度数据数组列表。
- showtickprefix，显示刻度前缀数据数组列表。
- showticksuffix，显示刻度后缀数据数组列表。

- side，边缘参数。
- spikecolor，峰值数据颜色。
- spikesides，峰值数据滑块。
- spikethickness，峰值数据厚度。
- tick0，刻度数据锚点。
- tickangle，刻度数据角度。
- tickcolor，刻度数据颜色。
- tickfont，刻度数据字体。
- tickformat，刻度数据格式。
- ticklen，刻度数据长度。
- tickmode，刻度数据模式。
- tickprefix，刻度数据前缀。
- ticks，刻度数据显示模式，包括 outside（刻度外）、inside（刻度内）、""（空字符串，无刻度）三种模式。
- ticksuffix，刻度数据后缀。
- ticktext，刻度数据文本。
- ticktextsrc，刻度数据文本数组列表。
- tickvals，刻度数据参数值数组列表。
- tickvalssrc，刻度数据参数值数据源数组列表。
- tickwidth，刻度数据宽度。
- title，标题。
- titlefont，标题字体。
- type，类型。
- zeroline，零线参数。
- zerolinecolor，零线颜色。
- zerolinewidth，零线宽度。

9．Python 对象属性提取技巧

前面笔者介绍过使用 help 命令查看 Layout 函数的参数与属性，代码如下。

```
import plotly.graph_objs as pygo
```

```
    print(help( pygo.Layout))
```

Matplotlib 绘图模块库和其他 Python 模块库的内置函数，也可以使用这种方法查看有关函数的内置属性。

Layout 画面布局函数的 help 命令的输出信息很多，其中开头部分的代码如下。

```
    |  Valid attributes for 'layout' at path [] under parents ():
    |
    |      ['direction', 'yaxis', 'angularaxis', 'hiddenlabels', 'xaxis',
'images',
    |      'smith', 'mapbox', 'radialaxis', 'bargroupgap', 'sliders',
'titlefont',
    |      'geo', 'ternary', 'scene', 'dragmode', 'barmode', 'annotations',
    |      'calendar', 'hidesources', 'boxgroupgap', 'barnorm', 'legend',
    |      'boxmode', 'width', 'autosize', 'title', 'paper_bgcolor',
'margin',
    |      'hiddenlabelssrc', 'orientation', 'plot_bgcolor', 'height',
    |      'showlegend', 'font', 'shapes', 'separators', 'boxgap',
'hovermode',
    |      'bargap', 'updatemenus']
```

其中，两个中括号之间的部分就是 Layout 画面布局函数的属性和参数。

在获取这些文本信息后，需要进一步整理排序，这里面有一些一线工程的操作技巧，笔者一般按如下步骤操作。

（1）打开一个类似 UltraEdit 的文本编辑器，粘贴以上属性和参数，只需要两个中括号之间的文本部分。

（2）把其中的“,”（逗号）用回车符号（一般是“^p”）代替，每个参数占一行。

（3）把其他多余的符号，包括空格、制表符（“^t”）、单引号、中括号全部用空字符代替。

（4）用编辑软件的排序功能，按字母顺序重新排列函数的属性和参数。

（5）把排列好的函数属性和参数文本复制到文档手册中。

（6）对于复杂的属性和参数，可以再复制一份，到百度翻译网页，使用“中英双语对照”模式翻译，再把翻译文本复制到文档手册中。

第 3 章

Plotly 高级图形

3.1 时间序列

3.1.1 使用方法

Plotly 对时间序列的支持比较友好，既支持字符串格式，又支持日期/时间格式，而且使用起来相当简单，代码（见文件 timeSeries）如下。

```
import plotly as py
import plotly.graph_objs as go

from datetime import datetime

# ----------pre def
pyplt = py.offline.plot

# ----------code
x_datetime = [datetime(year=2013, month=10, day=4),
              datetime(year=2013, month=11, day=5),
              datetime(year=2013, month=12, day=6)]
```

```
    x_string = ['2013-10-04', '2013-11-05', '2013-12-06']

    trace_datetime = go.Scatter(x=x_datetime, y=[1, 3, 6],name='trace_
datetime')
    trace_string = go.Scatter(x=x_string, y=[2, 4, 7],name='trace_string')
    data = [trace_datetime, trace_string]
    pyplt(data, filename='tmp/timeSeries.html')
```

运行结果如图 3-1 所示。

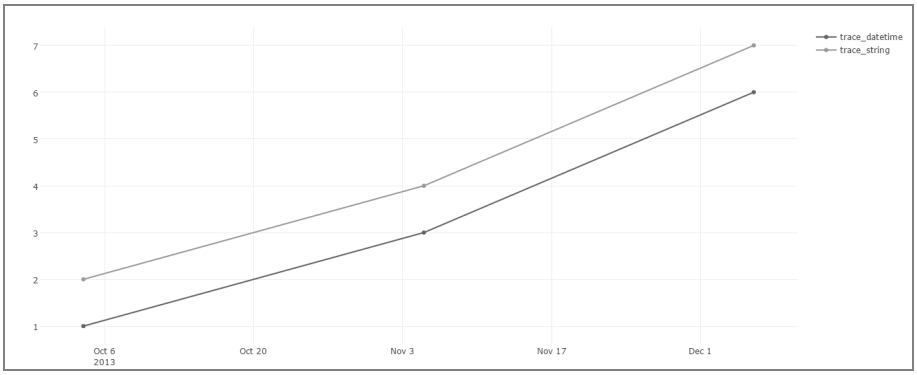

图 3-1　代码运行结果

从代码中可以看出，只要传入的 x 是日期或时间格式的字符串，Plotly 就会自动识别为时间格式。

3.1.2　时间范围约束

有时候，我们只想展示部分时间范围内的绘图结果，这时通过传递一个 range 参数即可实现这个功能，见文件 timeRange.py，代码如下。

```
    import plotly as py
    import plotly.graph_objs as go

    import datetime
```

```
    # ----------pre def
    pyplt = py.offline.plot

    # ----------code
    def to_unix_time(dt):
        '''
        :param dt:datetime 类型的时间戳
        :return: dt 相对于 utc 起始时间差别的毫秒数
        '''
        epoch =  datetime.datetime.utcfromtimestamp(0) # 获取 0 时刻对应的 utc
(世界标准时间)
        return (dt - epoch).total_seconds() * 1000 # 计算传入的时间相对于 utc
起始时间的毫秒数

    x = [datetime.datetime(year=2013, month=10, day=4),
        datetime.datetime(year=2013, month=11, day=5),
        datetime.datetime(year=2013, month=12, day=6)]
    data = [go.Scatter(
              x=x,
              y=[1, 3, 6])]

    layout = go.Layout(xaxis = dict(
                    range = [to_unix_time(datetime.datetime(2013, 10, 17)),
                             to_unix_time(datetime.datetime(2013, 11, 20))]
        ))

    fig = go.Figure(data = data, layout = layout)
    pyplt(fig, filename='tmp/timeRange.html')
```

输入结果如图 3-2 所示，可以看到，我们成功限制了输出结果的日期范围为 2013.10.17—2013.11.20。

在上面的代码中，关键部分的代码如下，主要用到了坐标轴参数（xaxis）里面的 range 参数，它是一个长度为 2 的列表，如果传递的数据类型为 datetime，那么其数值应该是 1970 年 1 月 1 日到传递时间的微秒数，to_unix_time 函数就是做这个事情的。

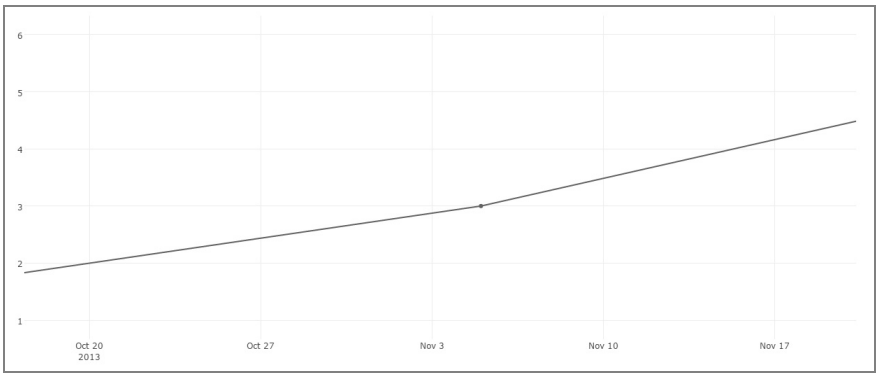

图 3-2　输入结果

```
    layout = go.Layout(xaxis = dict(
                    range = [to_unix_time(datetime.datetime(2013, 10, 17)),
                             to_unix_time(datetime.datetime(2013, 11, 20))]
        ))
```

如果想要恢复为默认的时间范围，可以单击界面右上角的 Autoscale 按钮，如图 3-3 示，就可以恢复成默认的时间范围。

图 3-3　Autoscale 按钮

代码运行结果如图 3-4 所示。

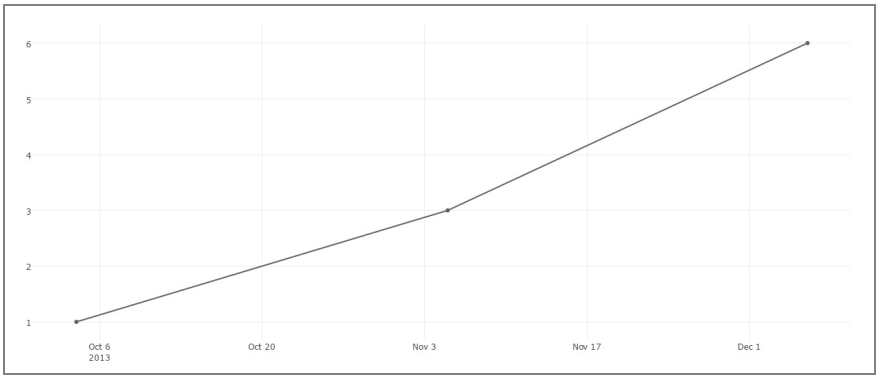

图 3-4　代码运行结果

3.2　滑动选择控件

对于金融时间序列的绘图，Pltoly 提供了强大的可定义模板，可以通过简单的

设置，轻松地实现图 3-5 和图 3-6 这两幅图中被圈出的功能。

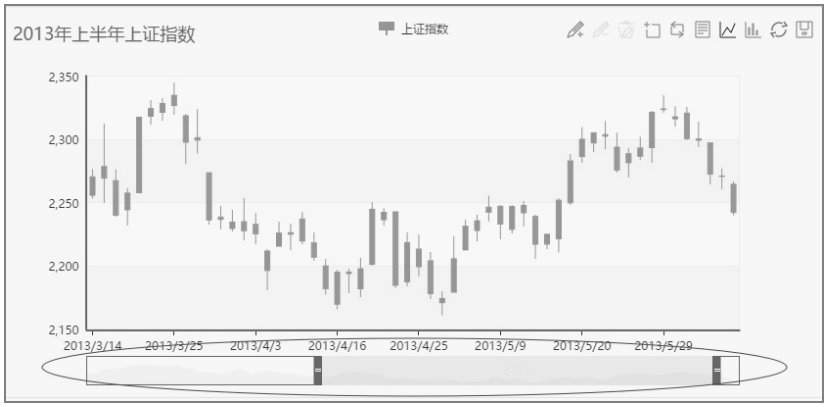

图 3-5　滑动选择控件 1

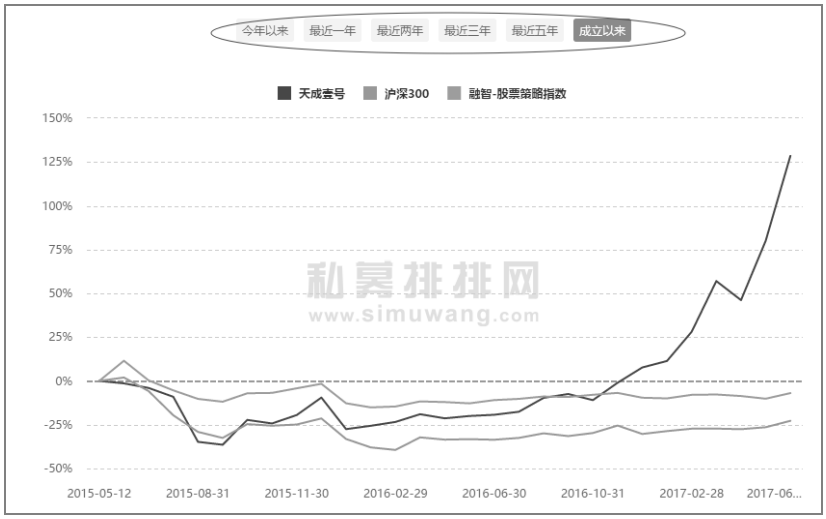

图 3-6　滑动选择控件 2

图 3-5 中是时间序列的滑块，可以通过拖动滑块左右临界点来调整时间范围；图 3-6 中是时间序列的选择器，可以通过单击不同的按钮来快速切换时间范围。这两种功能在 Plotly 中都可以通过相对简单的代码实现，代码如下（见文件 RangeSlide.py）。

```
import plotly as py
import plotly.graph_objs as go
```

```
import pandas as pd

# ----------pre def
pyplt = py.offline.plot

# ----------code
df = pd.read_csv(r'dat/day01.csv',index_col=[0])

trace = go.Scatter(x=df.index,
                   y=df.high)

data = [trace]
layout = dict(
    title='时间序列的滑块与选择器',
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label='1m',
                     step='month',
                     stepmode='backward'),
                dict(count=6,
                     label='6m',
                     step='month',
                     stepmode='backward'),
                dict(count=1,
                    label='YTD',#今天以来
                    step='year',
                    stepmode='todate'),
                dict(count=1,
                    label='1y',
                    step='year',
                    stepmode='backward'),
                dict(step='all')
            ])
        ),
        rangeslider=dict(),
```

```
        type='date'
    )
)
fig = dict(data=data, layout=layout)
pyplt(fig,filename='tmp/RangeSlide.html')
```

代码运行结果如图 3-7 所示。

图 3-7　代码运行结果

从图 3-7 可以看出，通过简单的代码就实现了时间序列滑块与选择器功能，而且效果相当漂亮，也非常好用。

滑块与选择器主要是通过 Layout 布局函数来实现的。对于选择器，主要在 xais.rangeselector.buttons 中定义一些参数，代码如下。

```
xaxis=dict(
      rangeselector=dict(
         buttons=list([
             dict(count=1,
                  label='1m',
                  step='month',
                  stepmode='backward'),
```

对上面的代码做如下解释。

（1）xaxis 表示对 *x* 轴进行定义；rangeselector 表示对 *x* 轴的选择器进行定义；buttons 表示选择器的按钮，是一个列表形式，每个列表元素都是一个选择器按钮。

（2）“count=1,step='month'”表示这个选择器覆盖的时间长度是“1 × month”，

即一个月。“label='1m'”表示选择器的标签为 1m。

（3）stepmode 有以下两种使用方法。

① 当 stepmode='backward'时，表示从后往前推进 count × step 时间。例如，当 count=1、step='year'时，表示时间范围将近一年。

② 当 stepmode='todate'时，一般只有一种用法，就是计算今年以来的时间。例如，当 count=1、step='year'时，表示时间范围从最后日期到最后日期所在的年初，而不是近一年。

3.3　表格

有时数据结构是一种表格结构，可以把这些数据展示出来，就像 Excel 显示出来的效果那样。幸运的是，Plotly 支持这种绘图方式，而且绘图效果非常美观。

3.3.1　入门案例

本案例见文件 table_first.py，用来说明创建表格的基本方法，代码如下。

```
import plotly as py
import plotly.figure_factory as FF

# ----------pre def
pyplt = py.offline.plot


data_matrix = [['国家', '年份', '人口'],
               ['中国',2000, 1267430000],
               ['美国', 2000, 282200000],
               ['加拿大', 2000, 27790000],
               ['中国', 2005, 1307560000],
               ['美国', 2005, 295500000],
               ['加拿大', 2005, 32310000],
               ['中国', 2010, 1340910000],
               ['美国', 2010, 309000000],
```

```
                    ['加拿大', 2010, 34000000]]

    table = FF.create_table(data_matrix)
    pyplt(table, filename='tmp\simple_table', show_link=False)
```

上面的代码没有太大的难点，有一点需要说明的是这段代码用到了figure_factory 类，这个类存储了除之前讲的基础绘图方法外的一些高级绘图方法，本节创建的表格使用的就是其中的高级绘图方法之一。

代码运行结果如图 3-8 所示。

国家	年份	人口
中国	2000	1267430000
美国	2000	282200000
加拿大	2000	27790000
中国	2005	1307560000
美国	2005	295500000
加拿大	2005	32310000
中国	2010	1340910000
美国	2010	309000000
加拿大	2010	34000000

图 3-8　代码运行结果

3.3.2　添加链接

Plotly 支持对文字的链接，本案例见文件 table_link.py，代码如下。

```
    import plotly as py
    import plotly.figure_factory  as FF


    # ----------pre def
    pyplt = py.offline.plot


    data_matrix = [['User', 'Language', 'Chart Type', '# of Views'],
                   ['<a href="https://plot.ly/~empet/folder/home">empet
</a>',
                    '<a href="https://plot.ly/python/">Python</a>',
```

```
                  '<a href="https://plot.ly/~empet/8614/">Network
Graph</a>',
                  298],
                 ['<a href="https://plot.ly/~Grondo/folder/home">Grondo
</a>',
                  '<a href="https://plot.ly/matlab/">Matlab</a>',
                  '<a href="https://plot.ly/~Grondo/42/">Subplots</a>',
                  356],
                 ['<a href="https://plot.ly/~Dreamshot/folder/home">
Dreamshot</a>',
                  '<a href="https://help.plot.ly/tutorials/">Web App</a>',
                  '<a href="https://plot.ly/~Dreamshot/6575/_2014-us-city-
populations/">Bubble Map</a>',
                  262],
                 ['<a href="https://plot.ly/~FiveThirtyEight/folder/home">
FiveThirtyEight</a>',
                  '<a href="https://help.plot.ly/tutorials/">Web App</a>',
                  '<a href="https://plot.ly/~FiveThirtyEight/30/">
Scatter</a>',
                  692],
                 ['<a href="https://plot.ly/~cpsievert/folder/home">
cpsievert</a>',
                  '<a href="https://plot.ly/r/">R</a>',
                  '<a href="https://plot.ly/~cpsievert/1130/">Surface
</a>',
                  302]]

    table = FF.create_table(data_matrix)
    pyplt(table, filename='tmp\linked_table', show_link=False)
```

代码运行结果如图 3-9 所示。

用户	编程语言	绘图类型	浏览数量
empet	Python	Network Graph	298
Grondo	Matlab	Subplots	356
Dreamshot	Web App	Bubble Map	262
FiveThirtyEight	Web App	Scatter	692
cpsievert	R	Surface	302

图 3-9　代码运行结果

单击第 1～第 3 列中的内容，会弹出一个窗口，打开相应的网页。从上面的代码中可以看出，实现链接的代码非常简单，只需要简单地运用 HTML 语言中<a> 标签的 href 属性即可，代码如下。

```
    '<a href="https://plot.ly/python/">Python</a>',
```

3.3.3 使用 Pandas

在 Python 语言中，进行数据分析离不开 Pandas。Pandas 最初是 AQR Capital Management 为解决数据分析任务而创建的一个模块，并在 2009 年开源。Pandas 功能强大且容易上手，在笔者的实际经验中，90%的数据分析任务都会交给 Pandas 完成。

同时，Pandas 的数据结构 DataFrame 又是一种表格结构，如果可以直接使用 Pandas 数据结构进行绘图，那么利用 Pandas 的数据处理优势，数据呈现将会变得非常简单。幸运的是，Plotly 的表格绘图支持 Pandas 的数据结构。

在 Plotly 中使用 Pandas 的代码如下（见文件 table_pandas.html）。

```
import plotly as py
from plotly.tools import FigureFactory as FF

import pandas as pd

# ----------pre def
pyplt = py.offline.plot

df = pd.read_csv(r'dat\day01.csv',index_col=[0])
df_sample = df[100:120]



table = FF.create_table(df_sample, index=True, index_title='Date')
pyplt(table, filename=r'tmp\table_pandas.html')
```

上面的代码看起来非常简单，只要把数据源换成 Pandas 的 DataFrame 数据格式即可（本例中为变量 df_sample）。

```
table = FF.create_table(df_sample, index=True, index_title='Date')
```

上面代码中的 index=True 表示在图表结果中显示第 1 列的索引列，默认情况下是不显示这个索引列的。

需要注意的是，这里只显示 20 行数据，这是因为 Plotly 表格数据渲染速度比较慢，如果数据量太大，会影响程序运行速度，并且在业界，这种网页表格的渲染过程需要分成好多页，每次渲染其中的一页。用于显示数据行数的代码如下。

```
df_sample = df[100:120]
```

代码运行结果如图 3-10 所示。

Date	open	high	close	low	volume	
2015-09-09	25.42	26.38	26.13	25.2	7117846.0	184577376.0
2015-09-08	24.7	25.5	25.39	24.15	6644897.0	164923216.0
2015-09-07	25.97	27.18	24.85	24.78	9742557.0	253932064.0
2015-09-02	24.56	26.26	26.14	24.5	12081475.0	309813440.0
2015-09-01	25.72	26.02	25.96	24.1	9999785.0	252966096.0
2015-08-31	25.68	26.1	26.06	24.65	8837602.0	223889808.0
2015-08-28	25.25	26.45	26.13	25.01	9475767.0	241693728.0
2015-08-27	24.05	24.9	24.89	23.49	7105518.0	172021568.0
2015-08-26	23.0	24.68	23.38	22.61	9140548.0	214603216.0
2015-08-25	23.5	25.29	22.6	22.6	12659454.0	295042272.0
2015-08-24	26.76	27.65	25.11	25.11	10978737.0	279062592.0
2015-08-21	29.7	30.39	27.9	27.86	7114918.0	205455376.0
2015-08-20	30.81	31.34	30.21	30.2	4735026.0	145880320.0
2015-08-19	29.75	31.54	31.47	29.01	9131096.0	277756384.0
2015-08-18	34.0	34.0	30.6	30.6	10338060.0	333824992.0
2015-08-17	32.25	34.0	34.0	32.04	11366881.0	378513696.0
2015-08-14	33.06	33.3	32.45	32.01	10404143.0	338391712.0
2015-08-13	32.95	32.97	32.91	31.93	7878838.0	255970160.0
2015-08-12	32.04	33.55	33.0	31.9	10056002.0	331154656.0

图 3-10　代码运行结果

3.3.4　改变大小与颜色

使用 Plotly 的 create_table 函数修改图表字体的大小与颜色，以及图表的颜色非常方便，代码如下（见文件 table_style.py）。

```
import plotly as py
import plotly.figure_factory as FF
import pandas as pd

# ----------pre def
```

```
    pyplt = py.offline.plot

    df = pd.read_csv(r'dat\day01.csv',index_col=[0])
    df_sample = df[100:150]



    colorscale = [[0, '#4d004c'],[.5, '#f2e5ff'],[1, '#ffffff']]
    fontcolor=['#FCFCFC', '#00EE00', '#FF3030']

    table = FF.create_table(df_sample,index=True, colorscale=colorscale,
font_colors=fontcolor)
    table.layout.width=1000

    # 设置文本大小
    for i in range(len(table.layout.annotations)):
        table.layout.annotations[i].font.size = 10 + (i % 50)*0.2
    pyplt(table, filename=r'tmp\style_table', show_link=False)
```

代码运行结果如图 3-11 所示。

	open	high	close	low	volume	amount
2015-09-09	25.42	26.38	26.13	25.2	7117846.0	184577376.0
2015-09-08	24.7	25.5	25.39	24.15	6644897.0	164923216.0
2015-09-07	25.97	27.18	24.85	24.78	9742557.0	253932064.0
2015-09-02	24.56	26.26	26.14	24.5	12081475.0	309813440.0
2015-09-01	25.72	26.02	25.96	24.1	9999785.0	252966096.0
2015-08-31	25.68	26.1	26.06	24.65	8837602.0	223889808.0
2015-08-28	25.25	26.45	26.13	25.01	9475767.0	241693728.0
2015-08-27	24.05	24.9	24.89	23.49	7105518.0	172021568.0
2015-08-26	23.0	24.68	23.38	22.61	9140548.0	214603216.0
2015-08-25	23.5	25.29	22.6	22.6	12659454.0	295042272.0
2015-08-24	26.76	27.65	25.11	25.11	10978737.0	279062592.0
2015-08-21	29.7	30.39	27.9	27.86	7114918.0	205455376.0
2015-08-20	30.81	31.34	30.21	30.2	4735026.0	145880320.0
2015-08-19	29.75	31.54	31.47	29.01	9131096.0	277756384.0
2015-08-18	34.0	34.0	30.6	30.6	10338060.0	333824992.0
2015-08-17	32.25	34.0	34.0	32.04	11366881.0	378513696.0
2015-08-14	33.06	33.3	32.45	32.01	10404143.0	338391712.0
2015-08-13	32.95	32.97	32.91	31.93	7878838.0	255970160.0

图 3-11　代码运行结果

上面的代码虽然简单，设计的内容却很复杂，以下几点是读者需要注意的地方。

（1）colorscale 参数是一个列表，默认是[[0, '#66b2ff'], [.5, '#d9d9d9'],[1, '#ffffff']]。[0, '#66b2ff']表示表头（第一行和有索引的第一列）的颜色是'#66b2ff'（深蓝色）；[.5, '#d9d9d9']表示除表头外，表格的奇数行的背景颜色是'#d9d9d9'（浅灰色）；同理，[1, '#ffffff']表示偶数行的背景色是'#ffffff'（白色），这种颜色方案的效果如图 3-11 所示。

（2）fontcolors 参数是一个列表，它既可以是一个元素的列表，表示所有表格的文字使用一种颜色（默认是['#000000']），也可以是三个元素的列表（如本案例），分别表示表头、奇数列、偶数列的文字颜色，甚至可以是一个与表格行数相同的列表，可以为每一行自定义一个颜色。

（3）下面的代码表示把表格的宽度设置为 1000。

```
table.layout.width=1000
```

（4）下面的代码表示为每一个单元格的文字设置字体大小。

```
# 设置文本大小
for i in range(len(table.layout.annotations)):
    table.layout.annotations[i].font.size = 10 + (i % 50)*0.2
```

以上代码本质上是在修改 Layout 的 annotations 注释函数。虽然看起来有些复杂，实际上做的事情却非常简单。以 i=0 为例（对应表格左上角的单元格），代码 table.layout.annotations[i].font.size = 10 + (i % 50)*0.2 的作用是把 annotations 由

```
{'align': 'left',
 'font': {'color': '#FCFCFC'},
 'showarrow': False,
 'text': '<b></b>',
 'x': -0.45,
 'xanchor': 'left',
 'xref': 'x1',
 'y': 0,
 'yref': 'y1'}
```

变成

```
{'align': 'left',
'font': {'color': '#FCFCFC', 'size': 10},
'showarrow': False,
'text': '<b></b>',
```

```
    'x': -0.45,
    'xanchor': 'left',
    'xref': 'x1',
    'y': 0,
    'yref': 'y1'}
```

也就是说，仅仅是把 font 关键字对应的字典{'color': '#FCFCFC'}变成了{'color': '#FCFCFC', 'size': 10}。

3.3.5 表格与图

在绘图时，有的数据适合以表格的形式展示出来，也有一些数据适合以图的形式展示出来，还有一些数据单独用表格或图都不能很好地展示，需要两者（表格和图）结合才能更好地展示出数据的特点。

本案例（见文件 table_subplot.py）提供了解决这个问题的方法，代码如下。

```
import plotly as py
import plotly.graph_objs as go
import plotly.figure_factory as FF


# ----------pre def
pyplt = py.offline.plot


# 添加表格数据
table_data = [['团队', '赢', '输', '平'],
              ['清华大学', 18, 4, 0],
              ['北京大学', 18, 5, 0],
              ['中国<br>人民大学', 16, 5, 0],
              ['复旦大学', 13, 8, 0],
              ['上海<br>交通大学', 13, 8, 0],
              ['同济大学', 13, 8, 0]]
# 通过 FF.create_table(table_data)来初始化一个 figure
figure = FF.create_table(table_data, height_constant=60)
```

```
# 添加绘图数据
teams = ['清华大学', '北京大学', '中国<br>人民大学',
        '复旦大学', '上海<br>交通大学', '同济大学']
scoreA = [3.54, 3.48, 3.0, 3.27, 2.83, 2.45]
scoreB = [2.17, 2.57, 2.0, 2.91, 2.57, 2.14]
# 对绘图添加 traces
trace1 = go.Scatter(x=teams, y=scoreA,
                marker=dict(color='#0099ff'),
                name='分值 A',
                xaxis='x2', yaxis='y2')
trace2 = go.Scatter(x=teams, y=scoreB,
                marker=dict(color='#404040'),
                name='分值 B',
                xaxis='x2', yaxis='y2')

# 把 trace 添加到 figure 中
figure['data'].extend(go.Data([trace1, trace2]))

# 设置 figure 的 layout
figure.layout.xaxis.update({'domain': [0, .5]})
figure.layout.xaxis2.update({'domain': [0.6, 1.]})
# 图的 yaxis 要与图的 xaxis 对应
figure.layout.yaxis2.update({'anchor': 'x2'})
figure.layout.yaxis2.update({'title': '分值'})
# 设置 figure 的边界
figure.layout.margin.update({'t':50, 'b':100})
figure.layout.update({'title': '20xx 年部分高校 xx 游戏比赛'})

# 画图
pyplt(figure, filename='tmp/subplot_table', show_link=False)
```

代码运行结果如图 3-12 所示。

图 3-12 中的文字内容是笔者为了更好地说明问题杜撰的，读者不要当真。这个案例中有几点需要说明。

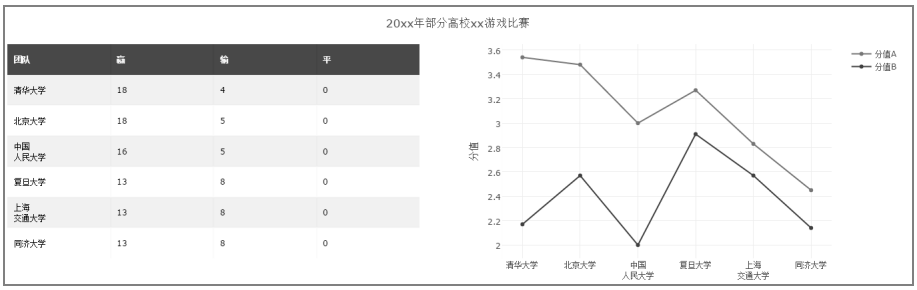

图 3-12　代码运行结果

（1）合并两幅图，本质上是对列表 figure['data']添加新的元素，代码如下。

```
# 把 trace 添加到 figure 中
figure['data'].extend(go.Data([trace1, trace2]))
```

（2）对两幅图进行布局管理，代码如下。

```
# 设置 figure 的 layout
figure.layout.xaxis.update({'domain': [0, .5]})
figure.layout.xaxis2.update({'domain': [0.6, 1.]})
```

（3）设置第二幅图的 y 轴与 x 轴对应。这样设置的结果是把字典 figure.layout.yaxis2 由空值变为{'anchor': 'x2', 'title': '分值'}，其中{'anchor': 'x2'}的意思是 yaxis2 坐标轴是与 xaxis2 坐标轴对应的。

```
# 图的 yaxis 要与图的 xaxis 对应
figure.layout.yaxis2.update({'anchor': 'x2'})
figure.layout.yaxis2.update({'title': '分值'})
```

（4）设置 figure 的边界与标题，这里设置距离顶部（top）50 个单位，距离底部（bottom）100 个单位，还可以对关键字 r（right）、l（left）进行设置。

```
# 设置 figure 的边界
figure.layout.margin.update({'t':50, 'b':100})
figure.layout.update({'title': '20xx 年部分高校 xx 游戏比赛'})
```

上面是水平布局的多子图绘图方式，有时候也需要进行垂直布局，相对于水平布局，垂直布局的代码只需要进行简单的修改（见文件 table_subplot2.py），代码如下。

```
import plotly as py
import plotly.graph_objs as go
```

```
import plotly.figure_factory as FF

# ----------pre def
pyplt = py.offline.plot


# 添加表格数据
table_data = [['团队', '赢', '输', '平'],
              ['清华大学', 18, 4, 0],
              ['北京大学', 18, 5, 0],
              ['中国<br>人民大学', 16, 5, 0],
              ['复旦大学', 13, 8, 0],
              ['上海<br>交通大学', 13, 8, 0],
              ['同济大学', 13, 8, 0]]
# 通过 FF.create_table(table_data)来初始化一个 figure
figure = FF.create_table(table_data, height_constant=60)


# 添加绘图数据
teams = ['清华大学', '北京大学', '中国<br>人民大学',
         '复旦大学', '上海<br>交通大学', '同济大学']
scoreA = [3.54, 3.48, 3.0, 3.27, 2.83, 2.45]
scoreB = [2.17, 2.57, 2.0, 2.91, 2.57, 2.14]


# 对绘图添加 traces
trace1 = go.Bar(x=teams, y=scoreA, xaxis='x2', yaxis='y2',
                marker=dict(color='#0099ff'),
                name='分值 A')
trace2 = go.Bar(x=teams, y=scoreB, xaxis='x2', yaxis='y2',
                marker=dict(color='#404040'),
                name='分值 B')


# 把 trace 添加到 figure 中
figure['data'].extend(go.Data([trace1, trace2]))
```

```
    # 设置 figure 的 layout
    figure.layout.yaxis.update({'domain': [0, .45]})
    figure.layout.yaxis2.update({'domain': [.6, 1]})
    # 图的 yaxis 要与图的 xaxis 对应
    figure.layout.yaxis2.update({'anchor': 'x2'})
    figure.layout.xaxis2.update({'anchor': 'y2'})
    figure.layout.yaxis2.update({'title': '分值'})
    # 设置 figure 的边界
    figure.layout.margin.update({'t':75, 'l':50})
    figure.layout.update({'title': '20xx 年部分高校 xx 游戏比赛'})
    # 重设 figure 的高度，因为两个图表如果垂直排列，原来的高度就不够，需要重设
    figure.layout.update({'height':800})

    # 画图
    pyplt(figure, filename='tmp/subplot_table_vertical.html',
show_link=False)
```

代码运行结果如图 3-13 所示。

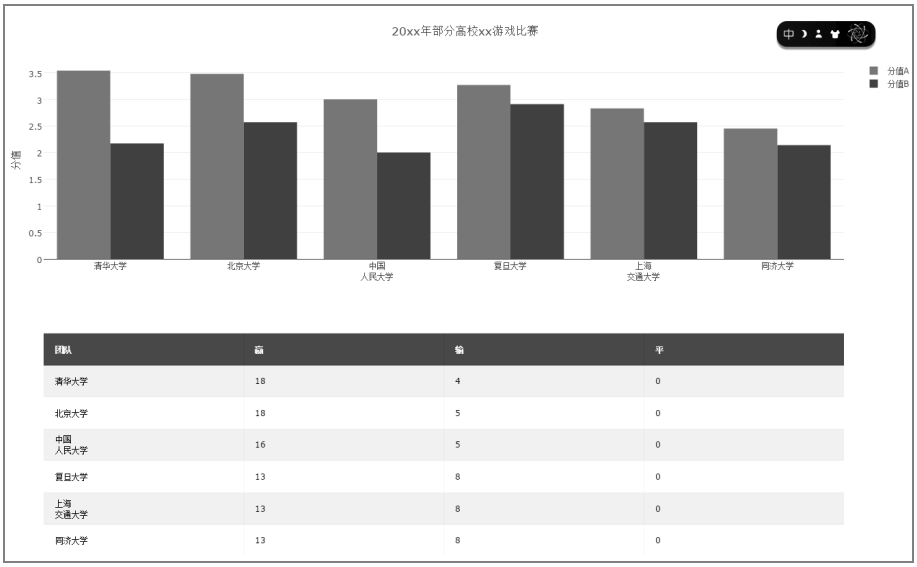

图 3-13　代码运行结果

3.4　多图表

所谓多图表，就是在一幅图中出现多个图表，这种呈现方式实际上在前面的章节中已经讲过，为了方便读者查阅这种呈现方式的使用方法，本节以一个简单的案例进行说明。

案例文件见 multi_plot.py，说明如何在一副图中呈现多个不同类型的图形，代码如下。

```
import plotly as py
import plotly.graph_objs as go

# ----------pre def
pyplt = py.offline.plot

x = list('ABCDEF')
trace1 = go.Scatter(
    x=x,
    y=[1.5, 1, 1.3, 0.7, 0.8, 0.9],
    name='line'
)
trace2 = go.Bar(
    x=x,
    y=[1, 0.5, 0.7, -1.2, 0.3, 0.4],
    name='bar'
)

data = [trace1, trace2]

layout = go.Layout(title='Bar-Line Demo',
                   xaxis=dict(
                       title='Keys',
                   ),
                   yaxis=dict(
                       title='Value',

                   ),
```

```
                    )

fig = dict(data=data, layout=layout)

pyplt(fig, filename='tmp/bar-line.html', show_link=False)
```

代码运行结果如图 3-14 所示。

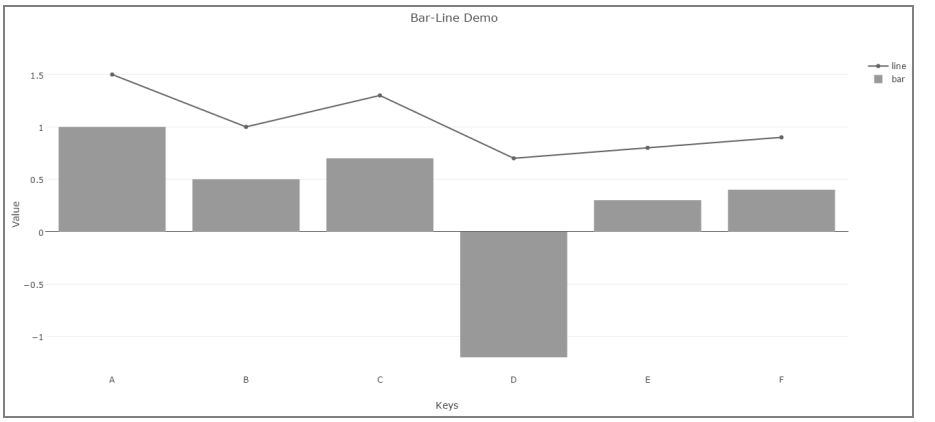

图 3-14　代码运行结果

3.5　多个坐标轴

有时，一幅图中多个图表的数值可能会相差很大，这样会导致数值较小的图表对坐标轴的变化不是很敏感，不能很好地体现图表的变化规律。因此，在绘图时就有多个坐标轴的需求。

3.5.1　双坐标轴

最简单的多个坐标轴就是双坐标轴了。使用 Plotly 绘制双坐标轴非常容易，本案例见文件 multi_yaxis.py，代码如下。

```
import plotly as py
import plotly.graph_objs as go
```

```
pyplt = py.offline.plot

trace1 = go.Scatter(
    x=[1, 2, 3],
    y=[40, 50, 60],
    name='yaxis 数据'
)
trace2 = go.Scatter(
    x=[2, 3, 4],
    y=[4, 5, 6],
    name='yaxis2 数据',
    yaxis='y2'
)
data = [trace1, trace2]
layout = go.Layout(
    title='Y 轴双轴示例',
    yaxis=dict(
        title='yaxis 标题'
    ),
    yaxis2=dict(
        title='yaxis2 标题',
        titlefont=dict(
            color='rgb(148, 103, 189)'
        ),
        tickfont=dict(
            color='rgb(148, 103, 189)'
        ),
        overlaying='y',
        side='right'
    )
)
fig = go.Figure(data=data, layout=layout)
plot_url = pyplt(fig, filename='tmp/multi_yaxis.html')
```

代码运行结果如图 3-15 所示。

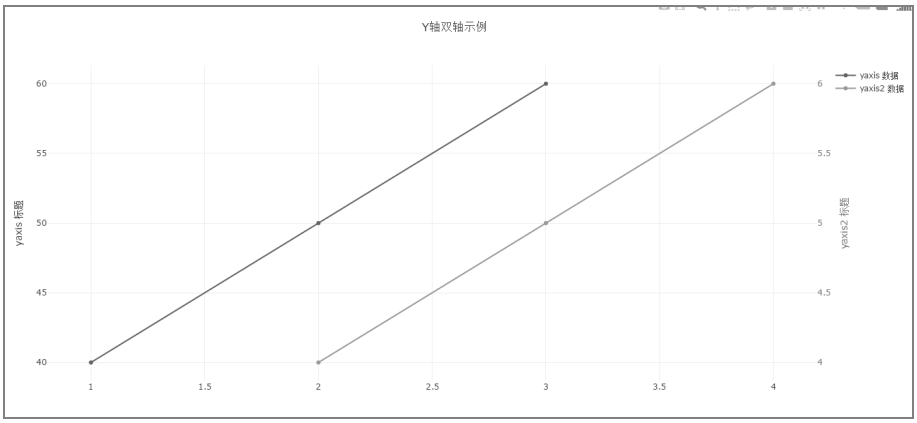

图 3-15　代码运行结果

在上面的代码中，比较关键的是下面几行代码：

```
titlefont=dict(
    color='rgb(148, 103, 189)'
),
tickfont=dict(
    color='rgb(148, 103, 189)'
),
overlaying='y',
side='right'
```

titlefont 用于设置标题格式，这里设置了 yaxis2 的标题颜色。

tickfont 用于设置刻度格式，这里设置了 yaxis2 的刻度颜色。

overlaying 是覆盖的意思，可以是"free"、"/^x([2-9]|[1-9][0-9]+)?$/"、"/^y([2-9]|[1-9][0-9]+)?$/"中的一种，本案例中参数为 y，表示会覆盖 *y* 轴。

side 的参数为 top、bottom、left、right 中的一种，表示坐标轴的位置。本案例中为 right，表示右坐标轴，即 yaxis2 为右坐标轴。

3.5.2　多坐标轴

相对于双坐标轴，Plotly 更让人激动的是可以实现多坐标轴的效果，比如 3 坐标轴、4 坐标轴。这些坐标轴都是在二维平面上实现的，绘制方法也很简单，代码

如下（见文件 multi_yaxis2.py）。

```
import plotly as py
import plotly.graph_objs as go

pyplt = py.offline.plot

trace1 = go.Scatter(
    x=[1, 2, 3],
    y=[4, 5, 6],
    name='yaxis1 数据'
)
trace2 = go.Scatter(
    x=[2, 3, 4],
    y=[40, 50, 60],
    name='yaxis2 数据',
    yaxis='y2'
)
trace3 = go.Scatter(
    x=[4, 5, 6],
    y=[40000, 50000, 60000],
    name='yaxis3 数据',
    yaxis='y3'
)
trace4 = go.Scatter(
    x=[5, 6, 7],
    y=[400000, 500000, 600000],
    name='yaxis4 数据',
    yaxis='y4'
)
data = [trace1, trace2, trace3, trace4]
layout = go.Layout(
    title='Y 轴多轴示例',
    width=800,
    xaxis=dict(
        domain=[0.3, 0.7]
    ),
    yaxis=dict(
```

```
            title='yaxis1 标题',
            titlefont=dict(
                color='#1f77b4'
            ),
            tickfont=dict(
                color='#1f77b4'
            )
        ),
        yaxis2=dict(
            title='yaxis2 标题',
            titlefont=dict(
                color='#ff7f0e'
            ),
            tickfont=dict(
                color='#ff7f0e'
            ),
            anchor='free',
            overlaying='y',
            side='left',
            position=0.15
        ),
        yaxis3=dict(
            title='yaxis3 标题',
            titlefont=dict(
                color='#d62728'
            ),
            tickfont=dict(
                color='#d62728'
            ),
            anchor='x',
            overlaying='y',
            side='right'
        ),
        yaxis4=dict(
            title='yaxis4 标题',
            titlefont=dict(
                color='#9467bd'
```

```
        ),
        tickfont=dict(
            color='#9467bd'
        ),
        anchor='free',
        overlaying='y',
        side='right',
        position=0.85
    )
)
fig = go.Figure(data=data, layout=layout)
plot_url = pyplt(fig, filename='tmp/multi_yaxis2.html')
```

代码运行结果如图 3-16 所示。

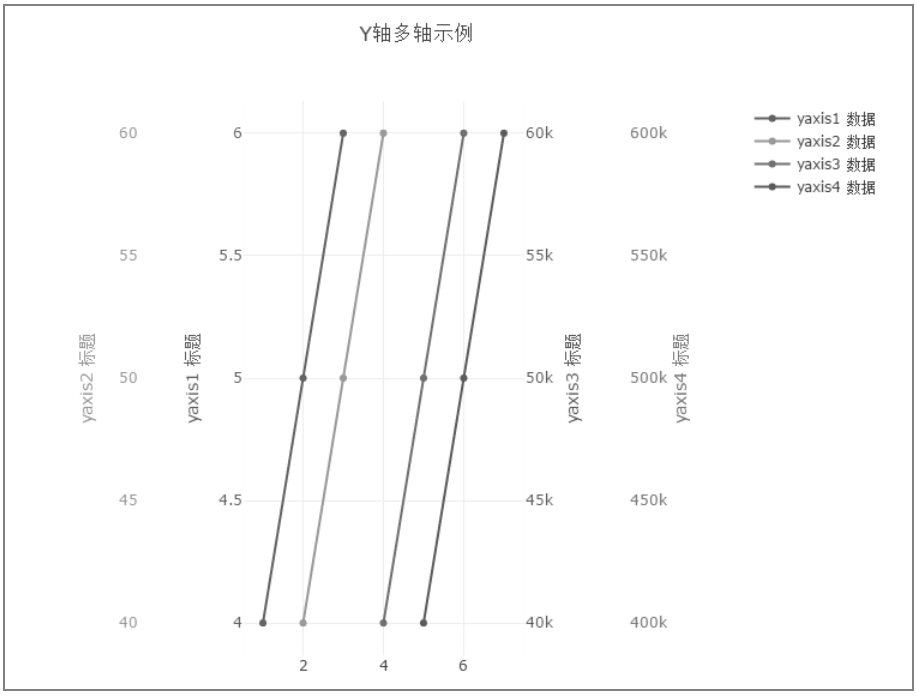

图 3-16　代码运行结果

可以看到，不同的坐标轴有不同的刻度及不同的颜色，很容易区分。

从代码上来看，实现上面的效果比较重要的是下面的几行代码。

```
    anchor='free',
    overlaying='y',
    side='left',
    position=0.15
```

anchor 的取值可以是 free、"/^x([2-9]|[1-9][0-9]+)?$/"、"/^y([2-9]|[1-9][0-9]+)?$/"，表示需要绑定的坐标轴。如果设置为 free，则其坐标轴由 position 的值指定，这里 position=0.15 表示在 0.15 这个位置的轴就是 yaxis2 所在的轴。

position 用来指定轴的位置，仅在 anchor='free'时该值有效。

3.5.3 共享坐标轴

有时会遇到这样的情况，*X* 轴的几个子图数值相差不大，*Y* 轴的几个子图数值相差较大，虽然可以像上例一样设置多个坐标轴，但是更想在一个坐标轴上呈现，所以就涉及到共享 *Y* 轴。本案例的代码如下（见文件 multi_share_yaxes.py）。

```
import plotly as py
import plotly.graph_objs as go

pyplt = py.offline.plot

trace1 = go.Scatter(
    x=[0, 1, 2],
    y=[10, 11, 12]
)
trace2 = go.Scatter(
    x=[2, 3, 4],
    y=[100, 110, 120],
    yaxis='y2'
)
trace3 = go.Scatter(
    x=[3, 4, 5],
    y=[1000, 1100, 1200],
    yaxis='y3'
)
```

```
data = [trace1, trace2, trace3]
layout = go.Layout(
    yaxis=dict(
        domain=[0, 0.33]
    ),
    legend=dict(
        traceorder='reversed'
    ),
    yaxis2=dict(
        domain=[0.33, 0.66]
    ),
    yaxis3=dict(
        domain=[0.66, 1]
    )
)
fig = go.Figure(data=data, layout=layout)
pyplt(fig, filename='tmp/multi_share_yaxes.html')
```

代码运行结果如图 3-17 所示。

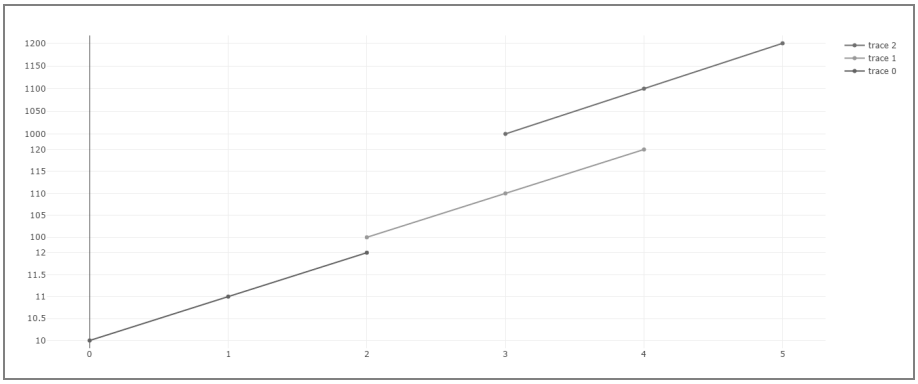

图 3-17　代码运行结果

从图 3-17 中可以看到，在一个图里呈现出了三个不同数字区域的图形。本案例主要用到了如下参数：

```
domain=[0.66, 1]
```

domain 参数用来指定坐标轴的区间范围，这里的设置为[0.66, 1]，表示 yaxis3

的区间为 0.66～1，所呈现出的结果为最上面的一条线。

可见，所谓共享 *Y* 坐标轴，是指不同的图表对整幅图在 *Y* 轴上的区域分割。

3.6 多子图

除多个坐标轴外，多个子图也是多个图表很好的呈现方式。

3.6.1 双子图（方法一）

最简单的多子图就属双子图了，为了方便在一幅图中呈现多个子图，Plotly 封装了一个很好的方法 make_subplots，这个方法的使用方式如下。

```
from plotly import tools
fig = tools.make_subplots(rows=2, cols=1)
```

make_subplots 的使用方法很简单，使用案例如下（见文件 subplot_simple.py）。

```
from plotly import tools
import plotly as py
import plotly.graph_objs as go


pyplt = py.offline.plot

trace1 = go.Scatter(
    x=[1, 2, 3],
    y=[4, 5, 6]
)
trace2 = go.Scatter(
    x=[20, 30, 40],
    y=[50, 60, 70],
)

fig = tools.make_subplots(rows=2, cols=1)

fig.append_trace(trace1, 1, 1)
```

```
fig.append_trace(trace2, 2, 1)

fig['layout'].update(height=600, width=600, title='i <3 subplots')
pyplt(fig, filename='tmp/subplot_simple.html')
```

代码运行结果如图 3-18 所示。

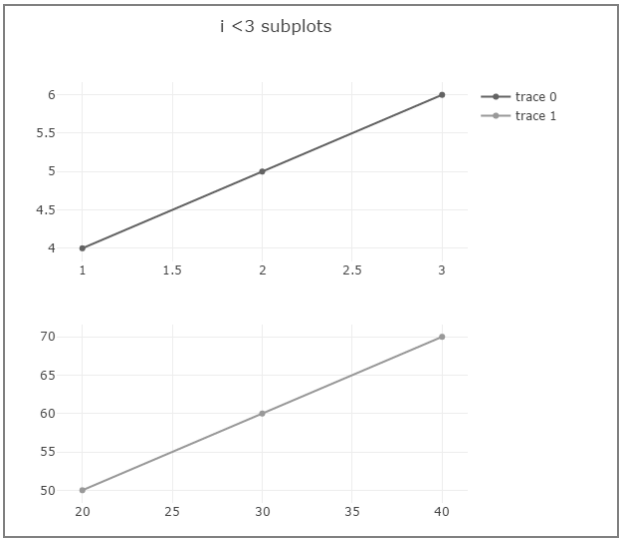

图 3-18　代码运行结果

3.6.2　双子图（方法二）

除上述方法外，在前面的内容中还使用过一种方法，那就是使用 domain 参数指定子图坐标轴在视图中的范围，这种方法的应用案例如下（见文件 subplot_simple2.py）。

```
import plotly as py
import plotly.graph_objs as go


pyplt = py.offline.plot

trace1 = go.Scatter(
```

```
    x=[1, 2, 3],
    y=[4, 5, 6]
)
trace2 = go.Scatter(
    x=[20, 30, 40],
    y=[50, 60, 70],
    xaxis='x2',
    yaxis='y2'
)
data = [trace1, trace2]
layout = go.Layout(
    xaxis=dict(
        domain=[0, 0.7]
    ),
    xaxis2=dict(
        domain=[0.8, 1]
    ),
    yaxis2=dict(
        anchor='x2'
    )
)
fig = go.Figure(data=data, layout=layout)
pyplt(fig, filename='tmp/subplot_simple2.html')
```

代码运行结果如图 3-19 所示。

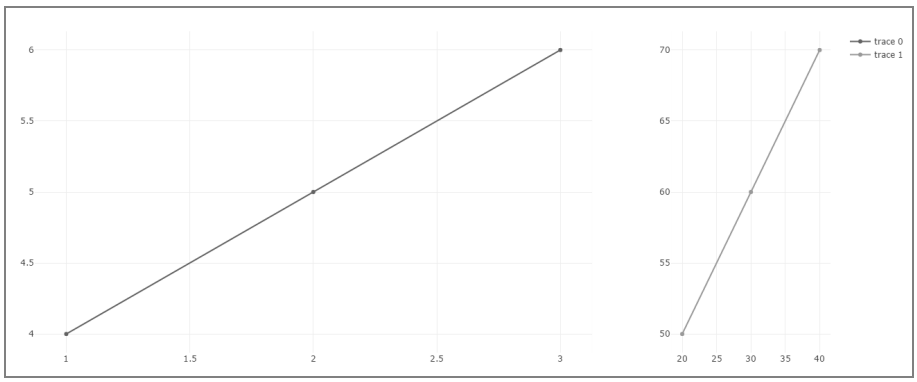

图 3-19　代码运行结果

3.6.3　多子图（方法一）

对于多个子图，绘制方法是一样的，以 make_subplots 方法为例（见文件 subplot_multi.py），案例代码如下。

```
    from plotly import tools
    import plotly as py
    import plotly.graph_objs as go

    pyplt = py.offline.plot

    trace1 = go.Scatter(x=[1, 2, 3],
                        y=[4, 5, 6],
                        mode='markers+text+lines',
                        text=['A', 'B', 'C'],
                        textposition='bottom'
                        )
    trace2 = go.Scatter(x=[20, 30, 40],y=[50, 60, 70])
    trace3 = go.Scatter(x=[300, 400, 500],
                        y=[600, 700, 800],
                        mode='markers+text+lines',
                        text=['D', 'E', 'F'],
                        textposition='bottom'
                        )
    trace4 = go.Scatter(x=[4000, 5000, 6000], y=[7000, 8000, 9000])

    fig = tools.make_subplots(rows=2, cols=2, subplot_titles=('Plot 1',
'Plot 2',
                                                              'Plot 3', 'Plot 4'))

    fig.append_trace(trace1, 1, 1)
    fig.append_trace(trace2, 1, 2)
    fig.append_trace(trace3, 2, 1)
    fig.append_trace(trace4, 2, 2)

    fig['layout'].update(height=600, width=1000, title='Multiple
Subplots2')
```

```
pyplt(fig, filename='tmp/subplot_multi.html')
```

代码运行结果如图 3-20 所示。

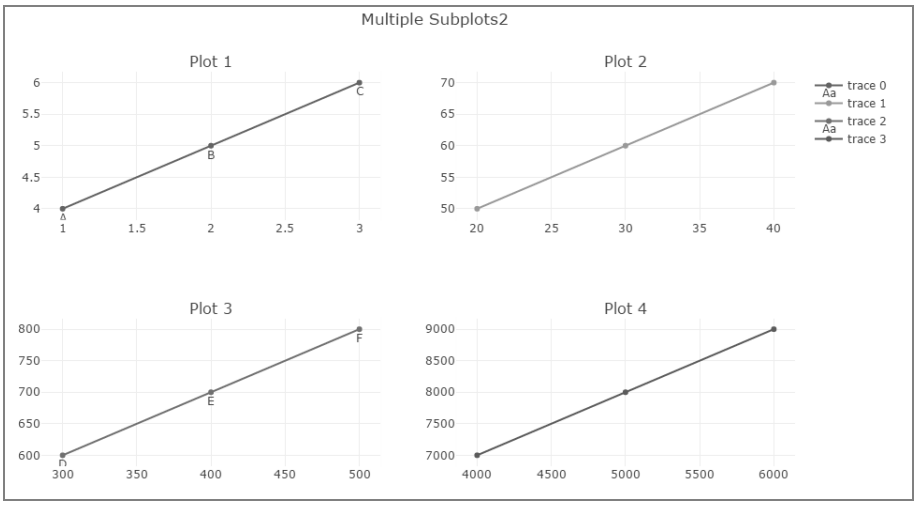

图 3-20　代码运行结果

3.6.4　多子图（方法二）

同样，也可以使用 domain 参数定义每个子图的位置范围（见文件 subplot_multi2.py），案例代码如下。

```
import plotly as py
import plotly.graph_objs as go


pyplt = py.offline.plot

trace1 = go.Scatter(
    x=[1, 2, 3],
    y=[4, 5, 6]
)
trace2 = go.Scatter(
```

```
    x=[20, 30, 40],
    y=[50, 60, 70],
    xaxis='x2',
    yaxis='y2'
)
trace3 = go.Scatter(
    x=[300, 400, 500],
    y=[600, 700, 800],
    xaxis='x3',
    yaxis='y3'
)
trace4 = go.Scatter(
    x=[4000, 5000, 6000],
    y=[7000, 8000, 9000],
    xaxis='x4',
    yaxis='y4'
)
data = [trace1, trace2, trace3, trace4]
layout = go.Layout(
    xaxis=dict(
        domain=[0, 0.45]
    ),
    yaxis=dict(
        domain=[0, 0.45]
    ),
    xaxis2=dict(
        domain=[0.55, 1]
    ),
    xaxis3=dict(
        domain=[0, 0.45],
        anchor='y3'
    ),
    xaxis4=dict(
        domain=[0.55, 1],
        anchor='y4'
    ),
    yaxis2=dict(
```

```
        domain=[0, 0.45],
        anchor='x2'
    ),
    yaxis3=dict(
        domain=[0.55, 1]
    ),
    yaxis4=dict(
        domain=[0.55, 1],
        anchor='x4'
    )
)
fig = go.Figure(data=data, layout=layout)
pyplt(fig, filename='tmp/subplots_multi2.html')
```

代码运行结果如图 3-21 所示。

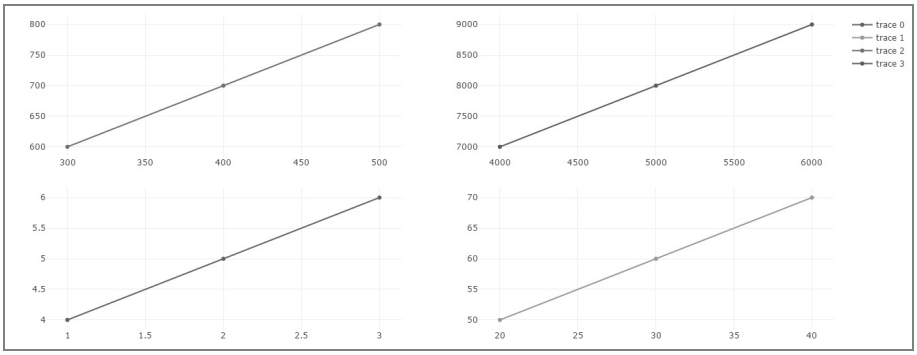

图 3-21　代码运行结果

3.6.5　分割视图区间

有时想为每个子图定义自己的大小，这就涉及到视图区间的分配问题。make_subplots 方法为解决这个问题专门封装了一个参数 specs，这个参数的简单使用方法如下（见文件 subplot_specs1.py）。

```
from plotly import tools
import plotly as py
import plotly.graph_objs as go
```

```
    pyplt = py.offline.plot


    trace0 = go.Scatter(
        x=[1, 2],
        y=[1, 2]
    )
    trace1 = go.Scatter(
        x=[1, 2],
        y=[1, 2]
    )
    trace2 = go.Scatter(
        x=[1, 2, 3],
        y=[2, 1, 2]
    )
    fig = tools.make_subplots(rows=2, cols=2, specs=[[{}, {}], [{'colspan':
2}, None]],
                              subplot_titles=('First Subplot','Second Subplot',
'Third Subplot'))

    fig.append_trace(trace0, 1, 1)
    fig.append_trace(trace1, 1, 2)
    fig.append_trace(trace2, 2, 1)

    fig['layout'].update(showlegend=False, title='Specs with Subplot
Title')
    pyplt(fig, filename='tmp/subplot_specs1.html')
```

代码运行结果如图 3-22 所示。

specs 参数定义了如何分配视图区间，本案例中的“specs=[[{}, {}], [{'colspan': 2}, None]]”表示第一行的两个子图平均分配区间，第二行的第一个子图占据 2 列的区间，并且不存在第二个子图，分配的结果如图 3-22 所示。

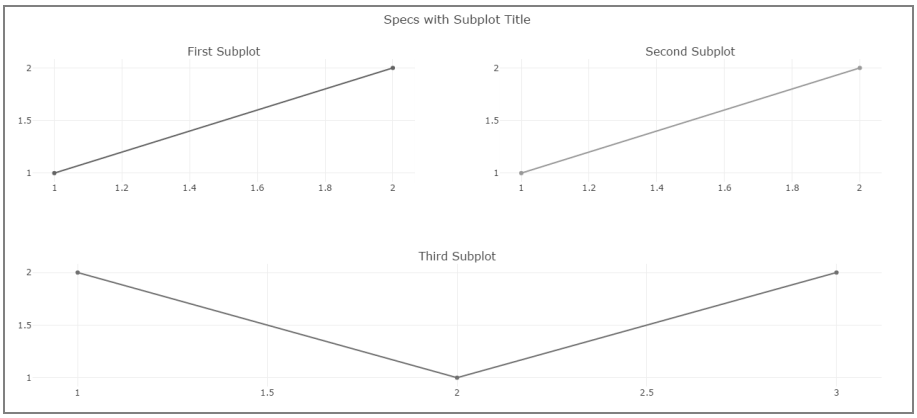

图 3-22　代码运行结果

本案例介绍了使用 make_subplots 方法分割视图，其实还可以分割得更复杂一些（见文件 subplot_specs2.py），代码如下。

```
from plotly import tools
import plotly as py
import plotly.graph_objs as go

pyplt = py.offline.plot

trace1 = go.Scatter(x=[1, 2], y=[1, 2], name='(1,1)')
trace2 = go.Scatter(x=[1, 2], y=[1, 2], name='(1,2)')
trace3 = go.Scatter(x=[1, 2], y=[1, 2], name='(2,1)')
trace4 = go.Scatter(x=[1, 2], y=[1, 2], name='(3,1)')
trace5 = go.Scatter(x=[1, 2], y=[1, 2], name='(5,1)')
trace6 = go.Scatter(x=[1, 2], y=[1, 2], name='(5,2)')

fig = tools.make_subplots(rows=5, cols=2,
                      specs=[[{}, {'rowspan': 2}],
                             [{}, None],
                             [{'rowspan': 2, 'colspan': 2}, None],
                             [None, None],
                             [{}, {}]],
                      print_grid=True)
```

```
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig.append_trace(trace3, 2, 1)
fig.append_trace(trace4, 3, 1)
fig.append_trace(trace5, 5, 1)
fig.append_trace(trace6, 5, 2)

fig['layout'].update(height=600, width=1000, title='specs examples')
pyplt(fig, filename='tmp/subplot_specs2.html')
```

代码运行结果如图 3-23 所示。

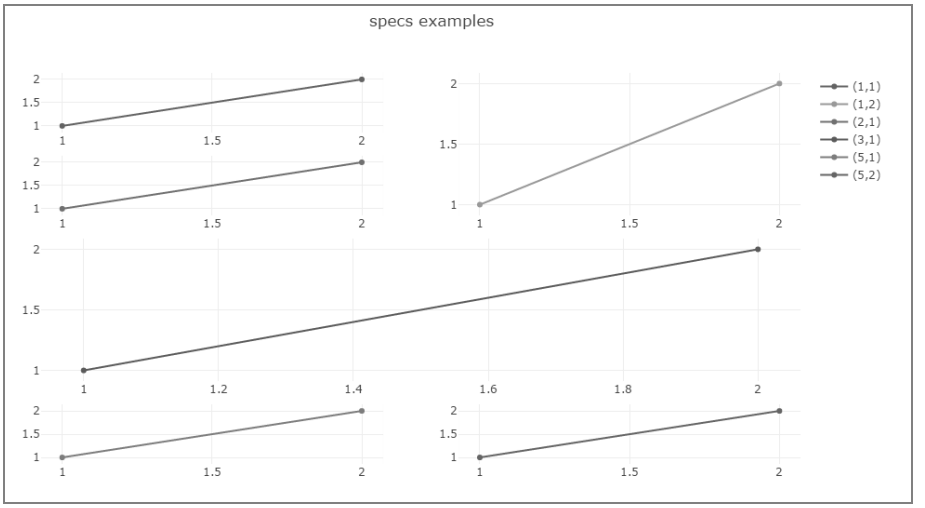

图 3-23　代码运行结果

本案例展示了把 5×2 的视图区间分配给 6 个子图，更复杂的图形也可以通过这种方式进行分配。

3.6.6　子图共享坐标轴（方法一）

有时多个子图的 X 坐标轴区间范围基本一致，从视觉效果来看，这些子图共同使用一个坐标轴效果会更好一些，这就涉及到了本节的内容。对于共享坐标轴，在之前的章节已经有所涉及，本节要讲的是如何在多个子图之间共享坐标轴。

在 Plotly 中，共享坐标轴很简单，用前面学过的知识就可以解决（见文件 subplot_share_axes.py），代码如下。

```
import plotly as py
import plotly.graph_objs as go


pyplt = py.offline.plot

trace1 = go.Scatter(
    x=[1, 2, 3],
    y=[2, 3, 4]
)
trace2 = go.Scatter(
    x=[20, 30, 40],
    y=[5, 5, 5],
    xaxis='x2',
    yaxis='y'
)
trace3 = go.Scatter(
    x=[2, 3, 4],
    y=[600, 700, 800],
    xaxis='x',
    yaxis='y3'
)
trace4 = go.Scatter(
    x=[4000, 5000, 6000],
    y=[7000, 8000, 9000],
    xaxis='x4',
    yaxis='y4'
)
data = [trace1, trace2, trace3, trace4]
layout = go.Layout(
    xaxis=dict(
        domain=[0, 0.45]
    ),
    yaxis=dict(
        domain=[0, 0.45]
    ),
```

```
        xaxis2=dict(
            domain=[0.55, 1]
        ),
        xaxis4=dict(
            domain=[0.55, 1],
            anchor='y4'
        ),
        yaxis3=dict(
            domain=[0.55, 1]
        ),
        yaxis4=dict(
            domain=[0.55, 1],
            anchor='x4'
        )
    )
    fig = go.Figure(data=data, layout=layout)
    pyplt(fig, filename='tmp/subplot_shared_axes.html')
```

代码运行结果如图 3-24 所示。

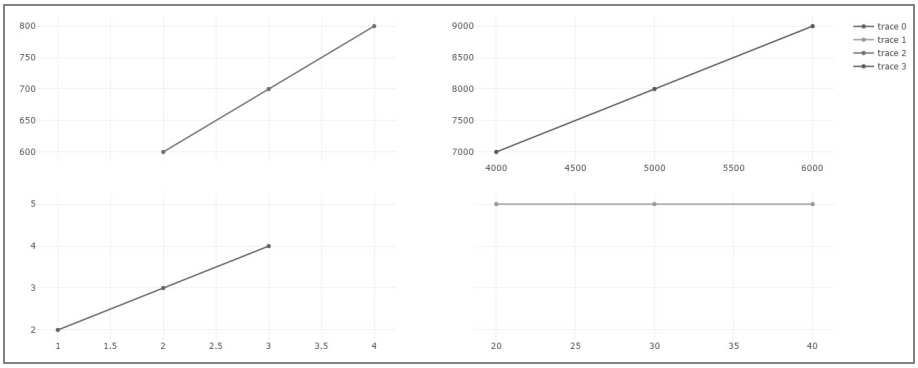

图 3-24　代码运行结果

3.6.7　子图共享坐标轴（方法二）

除上述方法外，make_subplots 也提供了很好的解决方法，通过设置参数 shared_yaxes 和 shared_xaxes 就可以控制共享坐标轴。

简单的案例如下，以共享 *Y* 坐标轴为例（见文件 subplot_share_yaxes.py），代码如下。

```
from plotly import tools
import plotly as py
import plotly.graph_objs as go

pyplt = py.offline.plot

trace0 = go.Scatter(
    x=[1, 2, 3],
    y=[2, 3, 4]
)
trace1 = go.Scatter(
    x=[20, 30, 40],
    y=[5, 5, 5],
)
trace2 = go.Scatter(
    x=[2, 3, 4],
    y=[600, 700, 800],
)
trace3 = go.Scatter(
    x=[4000, 5000, 6000],
    y=[7000, 8000, 9000],
)

fig = tools.make_subplots(rows=2, cols=2, shared_yaxes=True)

fig.append_trace(trace0, 1, 1)
fig.append_trace(trace1, 1, 2)
fig.append_trace(trace2, 2, 1)
fig.append_trace(trace3, 2, 2)

fig['layout'].update(height=600, width=600,
                     title='Multiple Subplots with Shared Y-Axes')
pyplt(fig, filename='tmp/subplot_share_yaxes.html')
```

代码运行结果如图 3-25 所示。

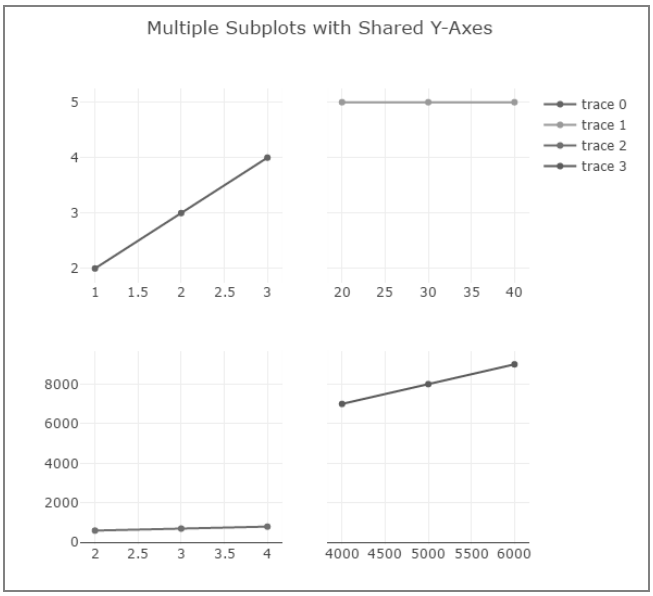

图 3-25　代码运行结果

当然，也可以设置两个坐标轴都共享，案例代码如下（见文件 subplot_share_xyaxes.py）。

```
from plotly import tools
import plotly as py
import plotly.graph_objs as go

pyplt = py.offline.plot

trace1 = go.Scatter(
    x=[0, 1, 2],
    y=[10, 11, 12]
)
trace2 = go.Scatter(
    x=[2, 3, 4],
    y=[100, 110, 120],
)
trace3 = go.Scatter(
    x=[3, 4, 5],
    y=[1000, 1100, 1200],
```

```
    )
    fig = tools.make_subplots(rows=3, cols=1, specs=[[{}], [{}], [{}]],
                              shared_xaxes=True, shared_yaxes=True,
                              vertical_spacing=0.001)
    fig.append_trace(trace1, 3, 1)
    fig.append_trace(trace2, 2, 1)
    fig.append_trace(trace3, 1, 1)

    fig['layout'].update(height=600, width=600, title='Stacked Subplots
with Shared X-Axes')
    pyplt(fig, filename='tmp/subplots_shared_xyaxes.html')
```

代码运行结果如图 3-26 所示。这个结果与 3.5.3 节共享坐标轴案例的结果一致，但是使用的是不同的方法。

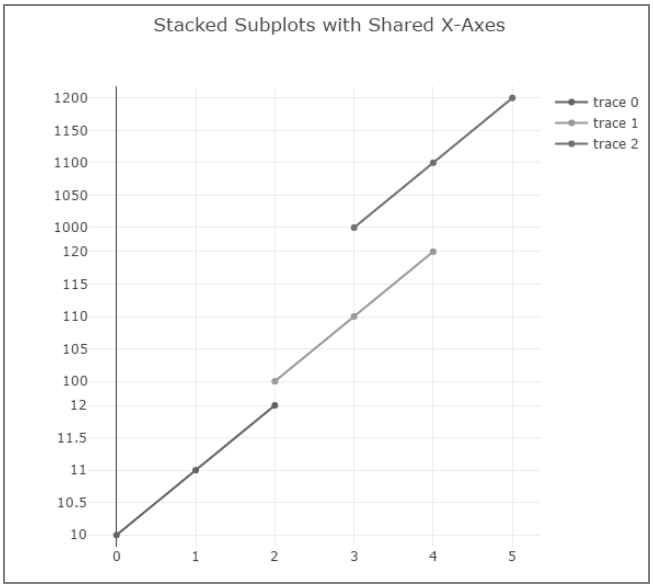

图 3-26　代码运行结果

3.6.8　子图坐标轴自定义

有时需要对每个子坐标轴进行自定义显示，这就需要用到本节的内容。在前面

的学习中我们知道，在 Plotly 中对已经定义好的绘图进行自定义显示，需要修改 figure 中的 layout 参数，在这里也不例外。

本案例代码如下（见文件 subplot_custom_axes.py）。

```
    from plotly import tools
    import plotly as py
    import plotly.graph_objs as go


    pyplt = py.offline.plot

    trace1 = go.Scatter(x=[1, 2, 3], y=[4, 5, 6])
    trace2 = go.Scatter(x=[20, 30, 40], y=[50, 60, 70])
    trace3 = go.Scatter(x=[300, 400, 500], y=[600, 700, 800])
    trace4 = go.Scatter(x=[4000, 5000, 6000], y=[7000, 8000, 9000])

    fig = tools.make_subplots(rows=2, cols=2, subplot_titles=('Plot 1',
'Plot 2', 'Plot 3', 'Plot 4'))
    fig.append_trace(trace1, 1, 1)
    fig.append_trace(trace2, 1, 2)
    fig.append_trace(trace3, 2, 1)
    fig.append_trace(trace4, 2, 2)

    fig['layout']['xaxis1'].update(title='xaxis 1 title')
    fig['layout']['xaxis2'].update(title='xaxis 2 title', range=[10, 50])
    fig['layout']['xaxis3'].update(title='xaxis 3 title', showgrid=False)
    fig['layout']['xaxis4'].update(title='xaxis 4 title', type='log')

    fig['layout']['yaxis1'].update(title='yaxis 1 title')
    fig['layout']['yaxis2'].update(title='yaxis 2 title', range=[40, 80])
    fig['layout']['yaxis3'].update(title='yaxis 3 title', showgrid=False)
    fig['layout']['yaxis4'].update(title='yaxis 4 title')

    fig['layout'].update(title='Customizing Subplot Axes')

    pyplt(fig, filename='tmp/subplot_custom_axes.html')
```

代码运行结果如图 3-27 所示。

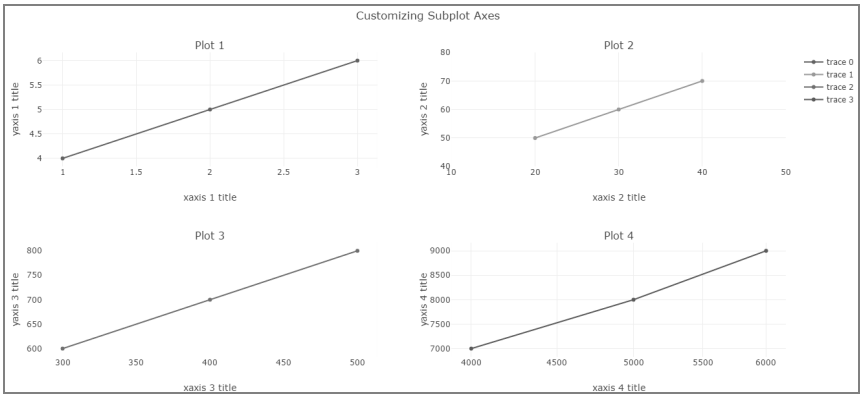

图 3-27　代码运行结果

本案例的代码只定义了坐标轴的显示范围、坐标轴类型和是否显示网格，对于其他的画面布局，也可以通过这种方式进行定义。

3.6.9　嵌入式子图

有时需要在一幅图上显示一个大图和一个小图，看起来就像小图嵌入到大图中一样。对于这个需求，Plotly 也可以满足。

显示的原理非常简单，就是利用 domain 参数，案例代码如下（见文件 subplot_insert.py）。

```
import plotly as py
import plotly.graph_objs as go


pyplt = py.offline.plot

trace1 = go.Scatter(
    x=[1, 2, 3],
    y=[4, 3, 2]
)
trace2 = go.Scatter(
```

```
    x=[20, 30, 40],
    y=[30, 40, 50],
    xaxis='x2',
    yaxis='y2'
)
data = [trace1, trace2]
layout = go.Layout(
    xaxis2=dict(
        domain=[0.6, 0.95],
        anchor='y2'
    ),
    yaxis2=dict(
        domain=[0.6, 0.95],
        anchor='x2'
    )
)
fig = go.Figure(data=data, layout=layout)
pyplt(fig, filename='tmp/subplot_insert.html')
```

代码运行结果如图 3-28 所示。

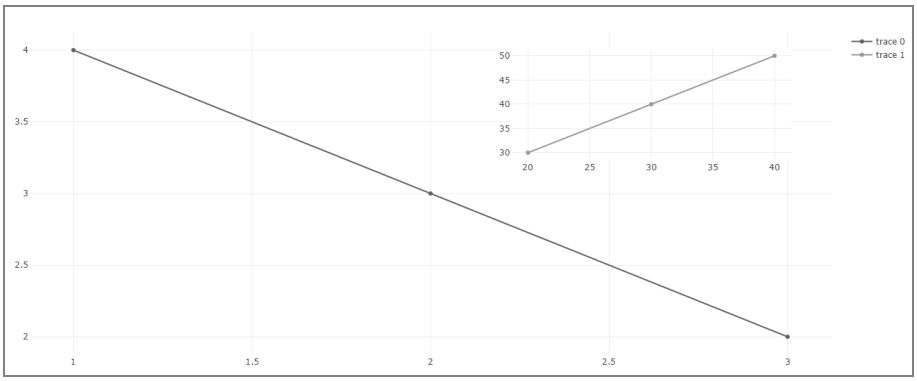

图 3-28　代码运行结果

3.6.10　混合图

下面介绍多图组合，本案例绘图的呈现效果特别的炫耀，先看一下炫酷的效果，

如图 3-29 所示。

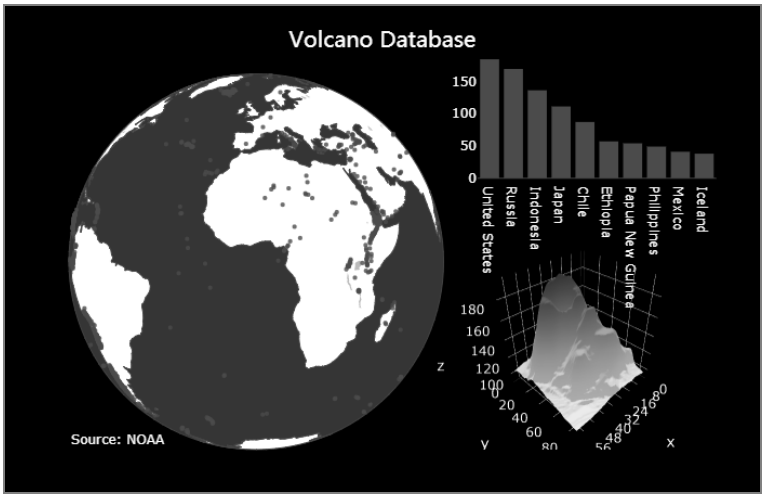

图 3-29　混合图效果

对于源代码，由于 3D 绘图的内容本书并没有涉及到，因此这里就不进行解读了，对 3D 图感兴趣的读者可以去官方网站学习相关的知识。本案例的代码如下（见文件 subplot_mixed.py）。

```
from plotly.graph_objs import *
import plotly as py
import plotly.graph_objs as go


pyplt = py.offline.plot

import pandas as pd

# read in volcano database data
df = pd.read_excel(r'dat/volcano_db.xlsx')



# frequency of Country
freq = df
```

```
    freq = freq.Country.value_counts().reset_index().rename(columns=
{'index': 'x'})

    # plot(1) top 10 countries by total volcanoes
    locations = go.Bar(x=freq['x'][0:10],y=freq['Country'][0:10], marker=
dict(color='#CF1020'))

    # read in 3d volcano surface data
    df_v = pd.read_excel(r'dat/volcano.xlsx')

    # plot(2) 3d surface of volcano
    threed = Surface(z=df_v.values.tolist(), colorscale='Reds', showscale=
False)

    # plot(3)  scattergeo map of volcano locations
    trace3 = {
      "geo": "geo3",
      "lon": df['Longitude'],
      "lat": df['Latitude'],
      "hoverinfo": 'text',
      "marker": {
        "size": 4,
        "opacity": 0.8,
        "color": '#CF1020',
        "colorscale": 'Viridis'
      },
      "mode": "markers",
      "type": "scattergeo"
    }

    data = go.Data([locations, threed, trace3])

    # control the subplot below using domain in 'geo', 'scene', and 'axis'
    layout = {
      "plot_bgcolor": 'black',
      "paper_bgcolor": 'black',
      "titlefont": {
```

```
        "size": 20,
        "family": "Raleway"
    },
    "font": {
        "color": 'white'
    },
    "dragmode": "zoom",
    "geo3": {
      "domain": {
        "x": [0, 0.55],
        "y": [0, 0.9]
      },
      "lakecolor": "rgba(127,205,255,1)",
      "oceancolor": "rgb(6,66,115)",
      "landcolor": 'white',
      "projection": {"type": "orthographic"},
      "scope": "world",
      "showlakes": True,
      "showocean": True,
      "showland": True,
      "bgcolor": 'black'
    },
    "margin": {
      "r": 10,
      "t": 25,
      "b": 40,
      "l": 60
    },
    "scene": {"domain": {
        "x": [0.5, 1],
        "y": [0, 0.55]
      },
            "xaxis": {"gridcolor": 'white'},
            "yaxis": {"gridcolor": 'white'},
            "zaxis": {"gridcolor": 'white'}
            },
    "showlegend": False,
```

```
    "title": "<br>Volcano Database",
    "xaxis": {
      "anchor": "y",
      "domain": [0.6, 0.95]
    },
    "yaxis": {
      "anchor": "x",
      "domain": [0.65, 0.95],
      "showgrid": False
    }
  }

  annotations = { "text": "Source: NOAA",
                  "showarrow": False,
                  "xref": "paper",
                  "yref": "paper",
                  "x": 0,
                  "y": 0}

  layout['annotations'] = [annotations]

  fig = go.Figure(data=data, layout=layout)
  pyplt(fig, filename = "tmp/Mixed Subplots Volcano.html")
```

3.7　绘制SVG

SVG（Scalable Vector Graphics，可伸缩矢量图形）使用 XML 格式定义图形，主要用于绘制二维图像。可以通过编写代码来展示简单的或复杂的图形，例如直线、曲线等；也可以在矢量图形编辑程序中创建这些直线、曲线，然后让程序导出代码。

为什么要使用 SVG？主要有下面几个方面的原因。

- 在浏览器中改变尺寸，SVG 图形质量不会有所损失。
- 相比其他位图，SVG 图形文件更小，可压缩性更强。
- SVG 可以被记事本等阅读器、搜索引擎访问。
- SVG 图形中的文本是可选的，同时也是可复制的。

- SVG 图形可以与 DOM、CSS 和 JavaScript 交互。
- SVG 可以制作成整体或局部动画。

虽然 SVG 有很多优点，但是本节并不是为了说明这些的，本节的主要任务是利用 SVG 的优点解决一些特殊的图形如圆、三角形、长方形等的绘制问题。

一般情况下，要绘制 SVG，就要通过 SVG 代码来实现，Plotly 对 SVG 进行了封装，可以用 Plotly 形式的语法来绘制 SVG。

3.7.1 线形图的绘制

绘制 SVG 最简单的就是绘制线形图，本案例代码如下（见文件 SVG_line.py）。

```
    import plotly as py
    import plotly.graph_objs as go

    # ----------pre def
    pyplt = py.offline.plot

    trace0 = go.Scatter(
        x=[1.2, 3.5, 5,0.5,7],
        y=[1, 1.8, 1.2,0.5,0.6],
        text=['垂直线(破折号)', '水平线（点+破折号）', '对角线（点）','对角线（相对于plot）','对角线（相对于轴）'],
        mode='text',
    )
    data = [trace0]
    layout = {
        'xaxis': {
            'range': [0, 8]
        },
        'yaxis': {
            'range': [0, 2.5]
        },
        'shapes': [
            # 画垂直线
            {
```

```
            'type': 'line',
            'x0': 1,
            'y0': 0,
            'x1': 1,
            'y1': 2,
            'line': {
                'color': 'rgb(55, 128, 191)',
                'width': 3,
                'dash':'dash',
            },
        },
        # 画水平线
        {
            'type': 'line',
            'x0': 2,
            'y0': 2,
            'x1': 5,
            'y1': 2,
            'line': {
                'color': 'rgb(50, 171, 96)',
                'width': 4,
                'dash': 'dashdot',
            },
        },
        # 画对角线
        {
            'type': 'line',
            'x0': 4,
            'y0': 0,
            'x1': 6,
            'y1': 2,
            'line': {
                'color': 'rgb(128, 0, 128)',
                'width': 4,
                'dash': 'dot',
            },
        },
```

```
        # 相对于轴绘图
        {
            'type': 'line',
            'xref': 'x',
            'yref': 'y',
            'x0': 4,
            'y0': 0,
            'x1': 8,
            'y1': 1,
            'line': {
                'color': 'rgb(55, 128, 191)',
                'width': 3,
            },
        },
        # 相对于 plot 绘图
        {
            'type': 'line',
            'xref': 'paper',
            'yref': 'paper',
            'x0': 0,
            'y0': 0,
            'x1': 0.5,
            'y1': 0.5,
            'line': {
                'color': 'rgb(50, 171, 96)',
                'width': 3,
            },
        },
    ]
}

fig = {
    'data': data,
    'layout': layout,
}

pyplt(fig, filename='tmp/SVG_line.html')
```

代码运行结果如图 3-30 所示。

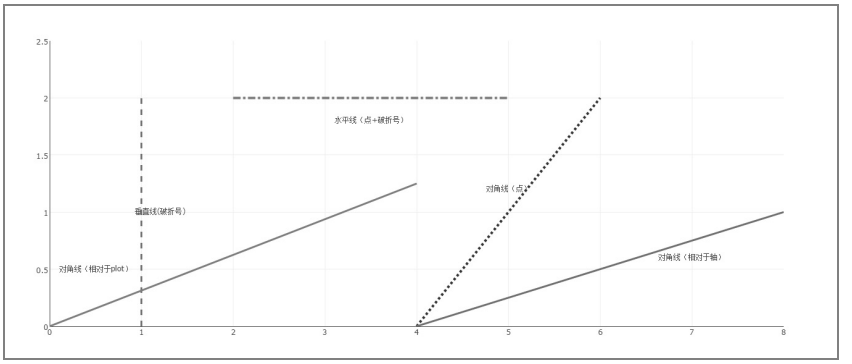

图 3-30　代码运行结果

前面讲解的图形都是通过 go.Scatter 等函数进行绘制的，绘制 SVG 却不一样，完全是在 layout 中进行设置的。上面的代码理解起来非常容易，就以其中一条垂直线（破折号）来说，只需要定义它的起点与终点的位置就可以了，而不用考虑中间位置的情况，代码如下。

```
    'x0': 1,
    'y0': 0,
    'x1': 1,
    'y1': 2,
```

对于本案例的代码，需要注意下面两点。

（1）dash 参数的使用。dash 参数用于设置线形图结果的呈现方式，可选值有 solid（本意是固体，这里指线条）、dot（点）、dash（破折号）、longdash（长破折号）、dashdot（点+破折号）和 longdashdot（长点+破折号），默认是 solid。

（2）xref（yref）参数的使用。该参数有两个值：paper 和 x（y）。如果值为 x，则 x0、x1 对应的值为坐标轴的绝对值，比如“x0=0,x1=4”表示线的起点 x0 在 0 处、终点 x1 在 4 处；如果值为 paper，则 x0、x1 对应的值为坐标轴的相对值，以本案例来说，“x0=0,x1=0.5”表示线的起点 x0 在 0 处、线的终点 x1 在 0.5×8=4 处。

3.7.2　线形图应用：创建图形的切线

本案例讲解如何为已有的图形添加切线，代码如下（见文件 SVG_tangent_line.py）。

```
import plotly as py
import plotly.graph_objs as go
import numpy as np

# ----------pre def
pyplt = py.offline.plot

x0 = np.linspace(1, 3, 200)
y0 = x0 * np.sin(np.power(x0, 2)) + 1

trace0 = go.Scatter(
    x=x0,
    y=y0,
)
data = [trace0]
layout = {
    'title': "f(x)=sin(x^2)+1",
    'shapes': [
        {
            'type': 'line',
            'x0': 1,
            'y0': 2.30756,
            'x1': 1.75,
            'y1': 2.30756,
            'opacity': 0.7,
            'line': {
                'color': 'red',
                'width': 2.5,
            },
        },
        {
            'type': 'line',
            'x0': 2.5,
            'y0': 3.80796,
            'x1': 3.05,
            'y1': 3.80796,
            'opacity': 0.7,
            'line': {
                'color': 'red',
```

```
                'width': 2.5,
            },
        },
        {
            'type': 'line',
            'x0': 1.90,
            'y0': -1.1827,
            'x1': 2.50,
            'y1': -1.1827,
            'opacity': 0.7,
            'line': {
                'color': 'red',
                'width': 2.5,
            },
        },
    ]
}
fig = {
    'data': data,
    'layout': layout,
}
pyplt(fig, filename='tmp/tangent-line.html')
```

代码运行结果如图 3-31 所示。

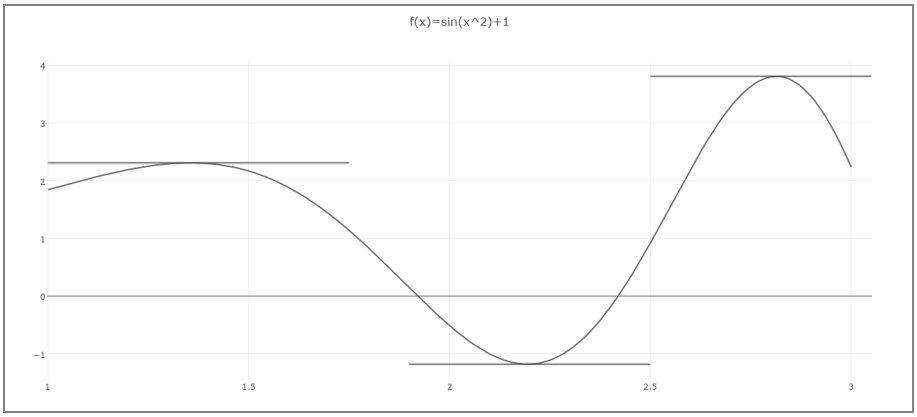

图 3-31　代码运行结果

上面代码的原理很简单，就是通过人工方法找出曲线 f(x)=sin(x^2)+1 在区间[1, 3]之间的几个极值点（区域最高点与区域最低点），然后根据这几个极值点绘制水平线。

3.7.3 矩形图的绘制

下面介绍矩形图的绘制，矩形图的绘制方法与线形图的绘制方法差不多，本案例代码如下（见文件 SVG_rectangle.py）。

```
    import plotly as py
    import plotly.graph_objs as go


    # ----------pre def
    pyplt = py.offline.plot


    trace0 = go.Scatter(
        x=[1.5, 4,1.5,4],
        y=[3.75, 3.75,2.2,2.2],
        text=['矩形', '矩形（填充）','矩形（相对于plot+填充）','矩形（相对于轴+填充）'],
        mode='text',
    )
    data = [trace0]
    layout = {
        'xaxis': {
            'range': [0, 8],
            'showgrid': False,
        },
        'yaxis': {
            'range': [0, 6.5]
        },
        'shapes': [
            # 普通的矩形
            {
```

```
            'type': 'rect',
            'x0': 1,
            'y0': 4,
            'x1': 2,
            'y1': 6,
            'line': {
                'color': 'rgba(128, 0, 128, 1)',
            },
        },
        # 填充颜色的矩形
        {
            'type': 'rect',
            'x0': 3,
            'y0': 4,
            'x1': 5,
            'y1': 6,
            'line': {
                'color': 'rgba(128, 0, 128, 1)',
                'width': 2,
            },
            'fillcolor': 'rgba(128, 0, 128, 0.7)',
        },
        # 相对于 plot 的矩形 + 填充颜色
        {
            'type': 'rect',
            'xref': 'paper',
            'yref': 'paper',
            'x0': 0.125,
            'y0': 0,
            'x1': 0.25,
            'y1': 0.3,
            'line': {
                'color': 'rgb(50, 171, 96)',
                'width': 3,
            },
            'fillcolor': 'rgba(50, 171, 96, 0.6)',
        },
```

```
        # 相对于轴的矩形 + 填充颜色
        {
            'type': 'rect',
            'xref': 'x',
            'yref': 'y',
            'x0': 3,
            'y0': 0,
            'x1': 5,
            'y1': 2,
            'line': {
                'color': 'rgb(55, 128, 191)',
                'width': 3,
            },
            'fillcolor': 'rgba(55, 128, 191, 0.6)',
        },

    ]
}
fig = {
    'data': data,
    'layout': layout,
}
pyplt(fig, filename='tmp/SVG_rectangle.html')
```

代码运行结果如图 3-32 所示。

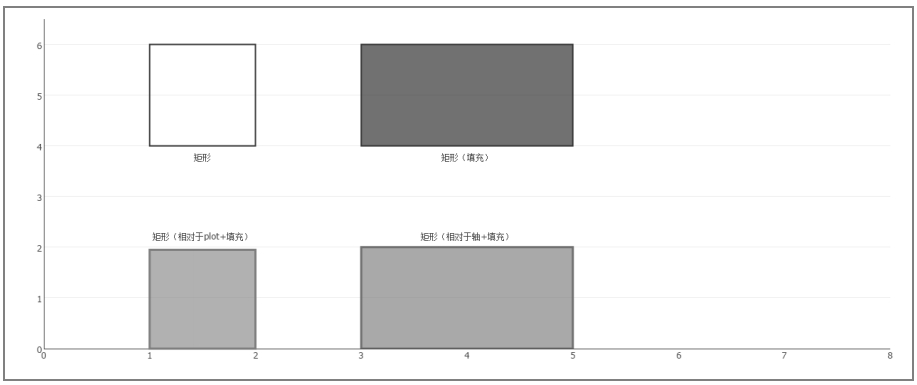

图 3-32　代码运行结果

与线形图的绘制案例一样，本案例同样提供了两种绘制方法，一种是一般的绘制方法（见图 3-32 的上半部分），另一种是有参考系的绘制方法（使用 xref、yref 参数，见图 3-32 的下半部分）。本案例的绘制方法总体上与线形图的绘制方法一样，有两点需要读者注意。

（1）type 参数的使用。参数值可以是 circle（圆）、rect（矩形）、path（路径）和 line（线），默认是 line。本案例使用的是 rect。

（2）fillcolor 参数的使用。设置填充的颜色，默认是“rgba(0,0,0,0)”（白色）。

3.7.4　矩形图应用：设置时间序列区域高亮显示

要实现时间序列区域的高亮显示，可以在原有的时间序列绘图基础上，通过 layout 布局函数绘制矩形，并用高亮的颜色填充，实现起来也非常简单（见文件 SVG_timestamp_highlight.py），代码如下。

```
    import plotly as py
    import plotly.graph_objs as go
    import pandas as pd

    # ----------pre def
    pyplt = py.offline.plot

    df = pd.read_csv(r'dat/day01.csv', index_col=['date'], parse_dates=
['date']) # 读取数据
    df.sort_index(inplace=True) # 将索引列从大到小排序
    df = df.iloc[-300:-100] # 选取其中的 200 行数据

    trace0 = go.Scatter(x=df.index, y=df['close'], mode='line', name=
'temperature')

    data = [trace0]
    layout = {
        # 通过创建矩形的方式来高亮显示某一个时间区间
        'shapes': [
            # 首先，高亮显示 2 月 4 日—3 月 6 日
```

```
        {
            'type': 'rect',
            # x参考系使用绝对坐标系（相对于轴）
            'xref': 'x',
            # y参考系使用相对坐标系（相对于plot）
            'yref': 'paper',
            'x0': '2015-02-04',
            'y0': 0,
            'x1': '2015-03-06',
            'y1': 1,
            'fillcolor': '#d3d3d3',
            'opacity': 0.2,
            'line': {
                'width': 0,
            }
        },
        # 其次，高亮显示区间5月20日—6月22日
        {
            'type': 'rect',
            'xref': 'x',
            'yref': 'paper',
            'x0': '2015-05-20',
            'y0': 0,
            'x1': '2015-06-22',
            'y1': 1,
            'fillcolor': '#d3d3d3',
            'opacity': 0.2,
            'line': {
                'width': 0,
            }
        }
    ]
}

fig = {'data': data, 'layout': layout}
pyplt(fig, filename='tmp/timestamp_highlight.html')
```

代码运行结果如图 3-33 所示。

图 3-33　代码运行结果

3.7.5　圆形图的绘制

下面介绍圆形图的绘制，绘制圆形图与绘制线形图、矩形图没有太大的区别，最大的不同是将 type 参数设置为 circle，本案例代码如下（见文件 SVG_circle.py）。

```
import plotly as py
import plotly.graph_objs as go

# ----------pre def
pyplt = py.offline.plot

trace0 = go.Scatter(
    x=[1.5, 3.5],
    y=[0.75, 2.5],
    text=['无填充圆',
          '有填充圆'],
    mode='text',
)
data = [trace0]

layout = {
    'xaxis': {
        'range': [0, 4.5],
        'zeroline': False,
```

```
    },
    'yaxis': {
        'range': [0, 4.5]
    },
    'width': 800,
    'height': 800,
    'shapes': [
        # 无填充圆
        {
            'type': 'circle',
            'xref': 'x',
            'yref': 'y',
            'x0': 1,
            'y0': 1,
            'x1': 3,
            'y1': 3,
            'line': {
                'color': 'rgba(50, 171, 96, 1)',
            },
        },
        # 有填充圆
        {
            'type': 'circle',
            'xref': 'x',
            'yref': 'y',
            'fillcolor': 'rgba(50, 171, 96, 0.7)',
            'x0': 3,
            'y0': 3,
            'x1': 4,
            'y1': 4,
            'line': {
                'color': 'rgba(50, 171, 96, 1)',
            },
        },
    ]
}
```

```
fig = {
    'data': data,
    'layout': layout,
}
pyplt(fig, filename='tmp/SVG_circle.html')
```

代码运行结果如图 3-34 所示。

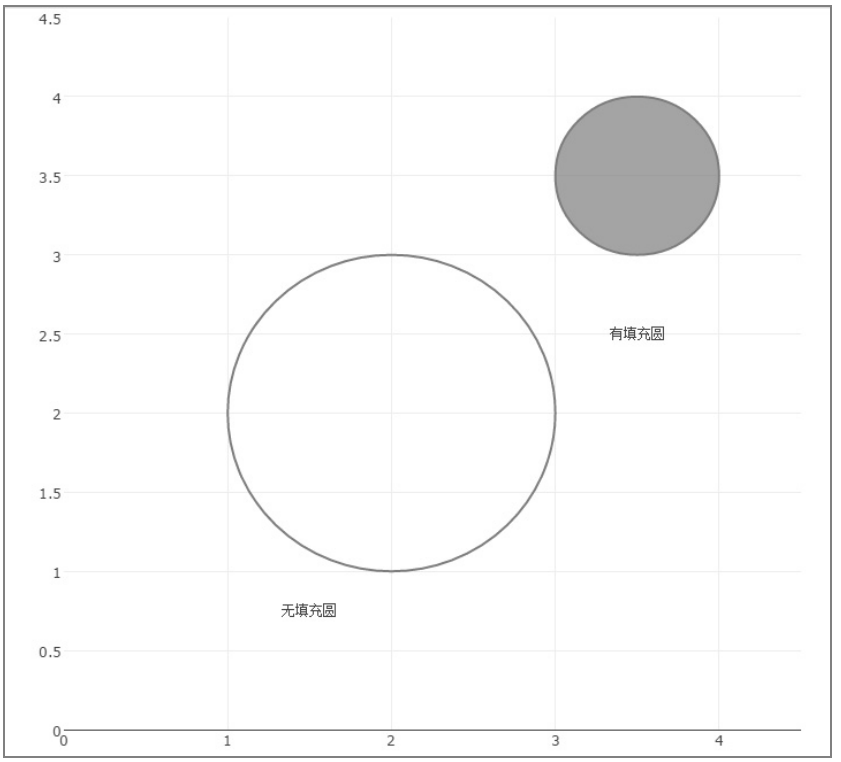

图 3-34　代码运行结果

注意

本案例中还设置了参数 hight=width=800,这个参数的意思读者可以这样理解：如果参数 hight != width，则按照我们自己的方式画出来的圆是一个椭圆，因为默认情况下 hight=450、width=750、hight != width，呈现出的结果就是一个椭圆。

3.7.6 圆形图应用：高亮显示散点图的聚集簇

下面简单介绍圆形图的应用，首先绘制几个散点图的聚集簇，然后以这些聚集簇为核心，绘制几个高亮显示的圆（见文件 SVG_clusters.py），代码如下。

```
import plotly as py
import plotly.graph_objs as go
import numpy as np

# ----------pre def
pyplt = py.offline.plot


x0 = np.random.normal(2, 0.45, 300)
y0 = np.random.normal(2, 0.45, 300)

x1 = np.random.normal(6, 0.4, 200)
y1 = np.random.normal(6, 0.4, 200)

x2 = np.random.normal(4, 0.3, 200)
y2 = np.random.normal(4, 0.3, 200)

trace0 = go.Scatter(
    x=x0,
    y=y0,
    mode='markers',
)
trace1 = go.Scatter(
    x=x1,
    y=y1,
    mode='markers'
)
trace2 = go.Scatter(
    x=x2,
    y=y2,
    mode='markers'
)
```

```
trace3 = go.Scatter(
    x=x1,
    y=y0,
    mode='markers'
)
layout = {
    'shapes': [
        {
            'type': 'circle',
            'xref': 'x',
            'yref': 'y',
            'x0': min(x0),
            'y0': min(y0),
            'x1': max(x0),
            'y1': max(y0),
            'opacity': 0.2,
            'fillcolor': 'blue',
            'line': {
                'color': 'blue',
            },
        },
        {
            'type': 'circle',
            'xref': 'x',
            'yref': 'y',
            'x0': min(x1),
            'y0': min(y1),
            'x1': max(x1),
            'y1': max(y1),
            'opacity': 0.2,
            'fillcolor': 'orange',
            'line': {
                'color': 'orange',
            },
        },
        {
            'type': 'circle',
```

```
            'xref': 'x',
            'yref': 'y',
            'x0': min(x2),
            'y0': min(y2),
            'x1': max(x2),
            'y1': max(y2),
            'opacity': 0.2,
            'fillcolor': 'green',
            'line': {
                'color': 'green',
            },
        },
        {
            'type': 'circle',
            'xref': 'x',
            'yref': 'y',
            'x0': min(x1),
            'y0': min(y0),
            'x1': max(x1),
            'y1': max(y0),
            'opacity': 0.2,
            'fillcolor': 'red',
            'line': {
                'color': 'red',
            },
        },
    ],
    'showlegend': False,
}
data = [trace0, trace1, trace2, trace3]
fig = {
    'data': data,
    'layout': layout,
}

pyplt(fig, filename='tmp/clusters.html')
```

代码运行结果如图 3-35 所示。

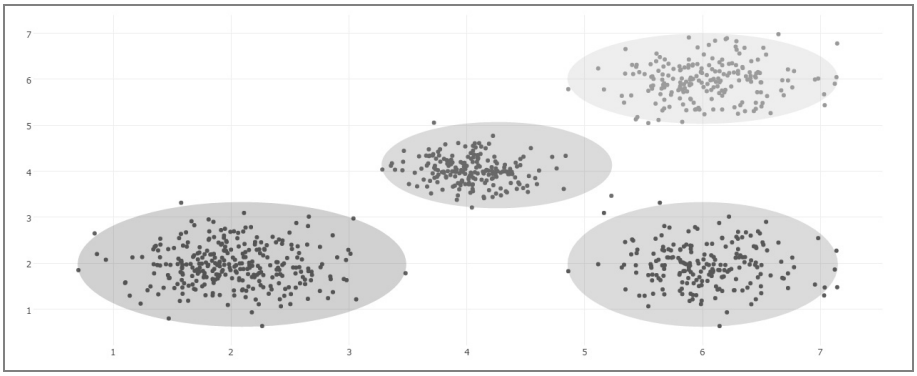

图 3-35　代码运行结果

第 4 章

Plotly 与 Pandas

Pandas 是 Python 语言的一个数据分析包，是由 AQR Capital Management 在 2008 年 4 月开发的，并于 2009 年年底开源，目前由专注于 Python 数据包开发的 PyData 开发组继续开发和维护，属于 PyData 项目的一部分。Pandas 最初是被作为金融数据分析工具而开发的，并为时间序列分析提供了很好的支持。从 Pandas 这个名称就可以看出，它是面板数据（Panel Data）和 Python 数据分析（Data Analysis）的结合。在 Pandas 出现之前，Python 数据分析的主力军只有 NumPy；在 Pandas 出现之后，它基本上占据了 Python 数据分析的霸主地位，在处理基础数据尤其是金融时间序列数据方面非常高效。

由于 Pandas 在数据分析中占有特殊地位，因此 Plotly 为 Pandas 绘图进行了专门的处理。

4.1 简单快速入门

在前面章节的案例中，已经或多或少使用过 Pandas。Pandas 使用起来很简单，Plotly 可以自动识别 Pandas 数据格式，下面介绍一些简单的案例。

4.1.1　基本线形图

由于 Plotly 可以识别 Pandas 数据格式，所以如果数据是 Pandas 的 DataFrame 或 Series 结构，就可以直接传递给 Plotly，不用转换成 list 数据格式。由于 Pandas 在数据读取与处理方面非常高效，所以 Plotly 与 Pandas 的无缝结合使绘图步骤更简单。以基本的线形图为例，代码如下（见文件 basic_line.py）。

```
import plotly as py  # 导入 Plotly 库并命名为 py
import plotly.graph_objs as go

import pandas as pd
import numpy as np

# -------------pre def
pyplt = py.offline.plot



N = 500
x = np.linspace(0, 1, N)
y = np.random.randn(N)
df = pd.DataFrame({'x': x, 'y': y})
df.head()

data = [
    go.Scatter(
        x=df['x'], # x 可以是 DataFrame 的某一列
        y=df['y']
    )
]

url = pyplt(data, filename='tmp/basic_line.html')
```

代码运行结果如图 4-1 所示。

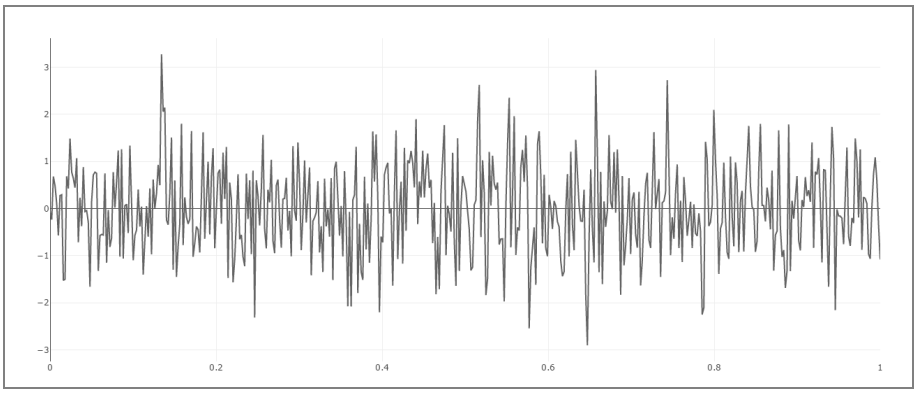

图 4-1　代码运行结果

4.1.2　基本散点图

散点图的绘制也非常简单，既可以用上一个案例中的 go.Scatter，也可以使用简单的字典格式（go.Scatter 本质上也是建立数据的字典对象）。本案例代码如下（见文件 basic_scatter.py）。

```
import plotly as py  # 导入 Plotly 库并命名为 py
import pandas as pd

# -------------pre def
pyplt = py.offline.plot

df_50 = pd.read_csv(r'dat/000016.csv', index_col=['date']) # 读取数据
df_50.sort_index(inplace=True) # 排序
df_50 = df_50.loc['2017-01-03':,:]
df_300 = pd.read_csv(r'dat/000300.csv', index_col=['date'])
df_300.sort_index(inplace=True) # 排序
df_300 = df_300.loc['2017-01-01':]

fig = {
    'data': [
        {
            'x': df_50.index,
```

```
            'y': df_50.volume,
            'mode': 'markers',
            'name': '上证 50'},
        {
            'x': df_300.index,
            'y': df_300.volume,
            'mode': 'markers',
            'name': '沪深 300'}
    ],
    'layout': {
        'xaxis': {'title': '时间',
                  # 'type': 'log',
                  },
        'yaxis': {'title': "成交量"}
    }
}

url = pyplt(fig, filename='tmp/basic_scatter.html')
```

代码运行结果如图 4-2 所示。

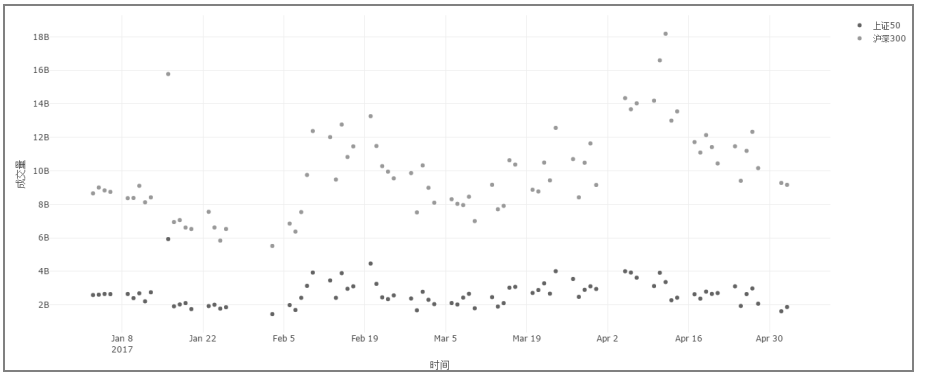

图 4-2　代码运行结果

4.1.3　基本柱状图

绘制基本的柱状图也非常简单（见文件 basic_bar.py），代码如下。

```
import plotly as py  # 导入 Plotly 库并命名为 py
import plotly.graph_objs as go

import pandas as pd
import numpy as np

# -------------pre def
pyplt = py.offline.plot

N = 40
x = np.linspace(0, 1, N)
y = np.random.randn(N)
df = pd.DataFrame({'x': x, 'y': y})
df.head()

data = [
    go.Bar(
        x=df['x'], # x 可以是 DataFrame 的某一列
        y=df['y']
    )
]

url = pyplt(data, filename=r'tmp/basic_bar.html')
```

代码运行结果如图 4-3 所示。

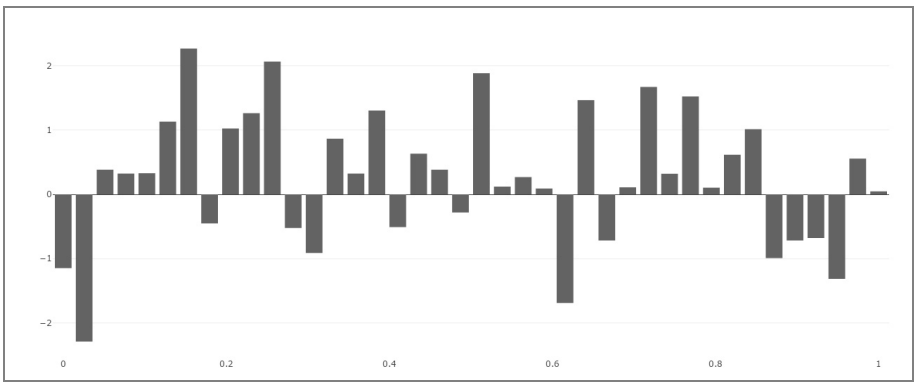

图 4-3　代码运行结果

4.2　使用cufflinks绘图

4.2.1　安装 cufflinks

本章一开始提过，为了方便绘图，Plotly 为 Pandas 做了特殊的封装，这个封装模块就是 cufflinks。cufflinks 的作用是改变 Pandas 绘图的默认呈现方式。Pandas 绘图默认的呈现方式是 Matplotlib，cufflinks 把这种呈现方式改为 Plotly。cufflinks 的安装方法很简单，用 pip 命令安装即可，代码如下。

```
pip install cufflinks
```

注意

需要注意的是，cufflinks 目前对 Pandas 的支持仅局限在 Jupyter Notebook 中，因此本节的代码全部是.ipynb 文件，读者需要打开 Jupyter Notebook 运行这些文件。

4.2.2　快速入门

使用 cufflinks 绘制线形图非常简单，与使用 Pandas 绘图并没有什么不同（见文件 cufflinks_demo.ipynb），代码如下。

```
import plotly.plotly as py
import cufflinks as cf
import pandas as pd

cf.set_config_file(offline=True, theme='ggplot')

N = 500
x = np.linspace(0, 1, N)
y = np.random.randn(N)
df = pd.DataFrame({'x': x, 'y': y})
df.set_index('x',inplace=True)

df.iplot(kind='scatter', filename='tmp/cf_line.html')
```

代码运行结果如图 4-4 所示。

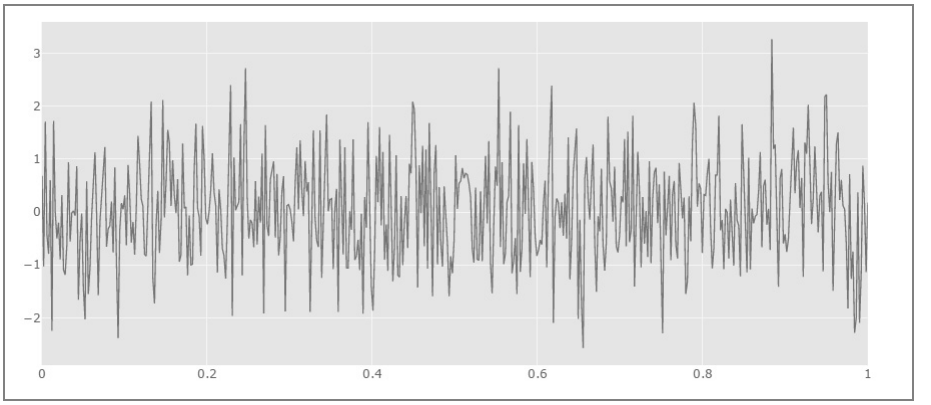

图 4-4　代码运行结果

cufflinks 并不影响 Pandas 的使用方式，只是在底层把 Pandas 绘图结果的呈现方式由 Matplotlib 变为 Plotly，我们平时怎么使用 Pandas，现在还怎么使用 Pandas。一般情况下，使用 cufflinks 改变 Pandas 的绘图呈现方式需要做如下几个步骤的工作。

（1）导入 cufflinks

导入 cufflinks 的代码如下。

```
import cufflinks as cf
```

（2）设置 Pandas 的输出模式

设置 Pandas 输出模式的代码如下。

```
cf.set_config_file(offline=True, theme='ggplot')
```

本案例使用离线绘图，并使用 R 语言中大名鼎鼎的 ggplot 主题。cufflinks 已经为我们封装好了多种非常流行的主题，包括 polar、pearl、henanigans、solar、ggplot、space 和 white 等，可以通过 cf.getThemes()方法获取当前版本所支持的所有主题。

注意

cf.set_config_file()是一个全局变量函数，也就是说这个函数会改变当前.ipynb 文件的所有 Pandas 绘图环境，所以在一个文件中，这个函数只出现一次就可以了。但是为了便于理解，在每个案例中都重新运行一下这个函数，这样做的好处是每个案例都是一个独立的个体，方便测试代码。

（3）绘制图形

绘制图形的代码如下。

```
df.iplot(kind='scatter', filename='tmp/cf_line.html')
```

cufflinks 只支持 iplot 的绘图函数，不支持 plot 的绘图函数，这点需要读者注意。我们可以通过输入“df.iplot?”查看其中的帮助文档，能够发现 iplot 函数与 Pandas 默认的“df.iplot?”帮助文档已经发生了根本性的改变，具体来说就是添加了好多 Plotly 的参数，如图 4-5 所示。

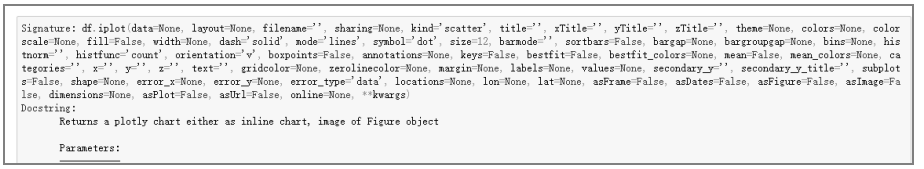
Signature: df.iplot(data=None, layout=None, filename='', sharing=None, kind='scatter', title='', xTitle='', yTitle='', zTitle='', theme=None, colors=None, colorscale=None, fill=False, width=None, dash='solid', mode='lines', symbol='dot', size=12, barmode='', sortbars=False, bargap=None, bargroupgap=None, bins=None, histnorm='', histfunc='count', orientation='v', boxpoints=False, annotations=None, keys=False, bestfit=False, bestfit_colors=None, mean=False, mean_colors=None, categories='', x='', y='', z='', text='', gridcolor=None, zerolinecolor=None, margin=None, labels=None, values=None, secondary_y='', secondary_y_title='', subplots=False, shape=None, error_x=None, error_y=None, error_type='data', locations=None, lon=None, lat=None, asFrame=False, asDates=False, asFigure=False, asImage=False, dimensions=None, asPlot=False, asUrl=False, online=None, **kwargs)
Docstring:
Returns a plotly chart either as inline chart, image of Figure object
Parameters:

图 4-5　帮助文档

4.2.3　快速获取数据

一般情况下，读者在学习数据处理与绘图的过程中，获取数据是一个很大的问题。有些读者的 NumPy 和 SciPy 水平比较高，可以通过 NumPy 和 SciPy 快速生成自己所需的数据；有些读者习惯使用外部数据，如通过 df=pd.read_csv()方式获取数据。本节将介绍一种更方便地获取数据的方法，那就是通过 cufflinks 快速生成所需要的数据，在学习 Pandas 数据处理与绘图的过程中非常高效。

快速获取数据案例的代码如下（见文件 cufflinks_demo.ipynb）。

```
import plotly.plotly as py
import cufflinks as cf
import pandas as pd

cf.set_config_file(offline=True, theme='polar')

df = cf.datagen.scatter() # 生成散点图数据



df.iplot(kind='scatter', mode='markers', x='x', y='y')
```

代码运行结果如图 4-6 所示。

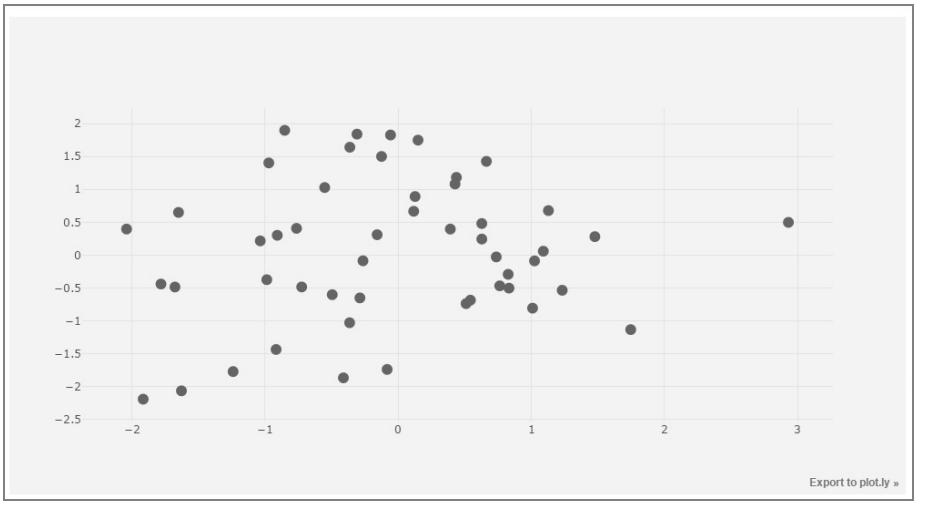

图 4-6　代码运行结果

这里主要介绍下面的函数：

```
df = cf.datagen.scatter() # 生成散点图数据
```

cf.datagen 是 cufflinks 封装好的生成 Pandas 数据的模块，里面包含了常见的绘图（如 bar、pie、scatter、ohlc）数据所需要的函数，本案例使用了 scatter 数据。读者可以通过 dir(cf.datagen)命令查看可用的数据。在绘图过程中使用这些数据非常方便。

4.2.4　自定义绘图

cufflinks 的绘图函数 df.iplot 功能强大，当然参数也比较多，本案例主要讲解其中常见且高效的一些参数的使用方法，以实现更丰富的绘图功能。

本案例的代码如下（见文件 cufflinks_demo.ipynb）。

```
import plotly.plotly as py
import cufflinks as cf
import pandas as pd
```

```
    cf.set_config_file(offline=True, world_readable=True)


    df = cf.datagen.lines(n_traces=3,columns=['one','two','three']) # 生成
一个 3 列的 DataFrame（默认 5 列）

    colors = ['red', 'blue', 'black'] # 自定义每条线的颜色
    dashes = ['solid', 'dash', 'dashdot'] # 自定义每条线的类型
    widths = [2, 4, 6] # 自定义每条线的宽度

    plot_url = df.iplot(kind='scatter', mode='lines', colors=colors,
dash=dashes,width=widths,xTitle='日期',yTitle='数量',title='自定义绘图')
```

代码运行结果如图 4-7 所示。

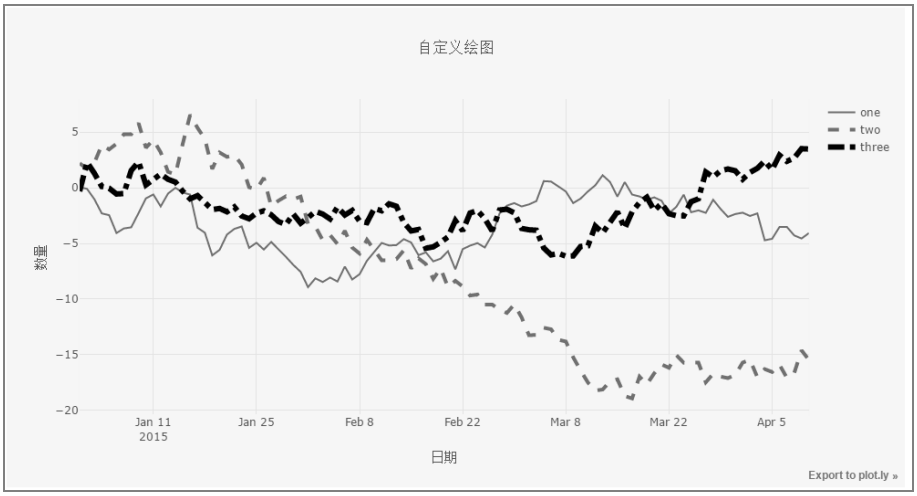

图 4-7　代码运行结果

上面的代码虽然简单，但是实现的功能还是挺强大的，实现了如下功能。

（1）自定义数据源，定义数据生成的长度和每条数据的名称

```
    df = cf.datagen.lines(n_traces=3,columns=['one','two','three']) # 生成
一个 3 列的 DataFrame（默认 5 列）
```

（2）自定义每条线的颜色、类型和宽度

```
    colors = ['red', 'blue', 'black'] # 自定义每条线的颜色
```

```
dashes = ['solid', 'dash', 'dashdot'] # 自定义每条线的类型
widths = [2, 4, 6] # 自定义每条线的宽度
```

（3）自定义图与坐标轴的标题

```
xTitle='日期',yTitle='数量',title='自定义绘图'
```

经过以上操作，基本上完成了一幅图的常见设置。

4.2.5 常见经典图形

前面主要介绍了线形图和散点图，接下来介绍其他常见的经典图形（见文件cufflinks_demo.ipynb），代码如下。

1. 柱状图

绘制柱状图的代码如下。

```
import plotly.plotly as py
import cufflinks as cf

cf.set_config_file(offline=True, world_readable=True, theme='ggplot')
df = cf.datagen.bars()
df.iplot(kind='bar', filename=r'tmp/cf_bar.html')
```

代码运行结果如图 4-8 所示。

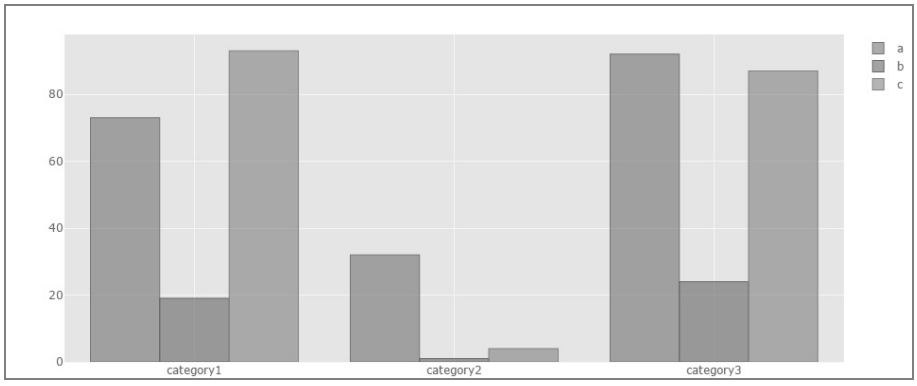

图 4-8 柱状图

2. 层叠柱状图

绘制层叠柱状图的代码如下。

```
    import plotly.plotly as py
    import cufflinks as cf



    cf.set_config_file(offline=False, world_readable=True, theme='ggplot')
    df = cf.datagen.bars(n_categories=4,columns=['A', 'B', 'C', 'D'])

    df.iplot(kind='bar', barmode='stack', filename='cufflinks/stacked-bar-
chart')
```

代码运行结果如图 4-9 所示。

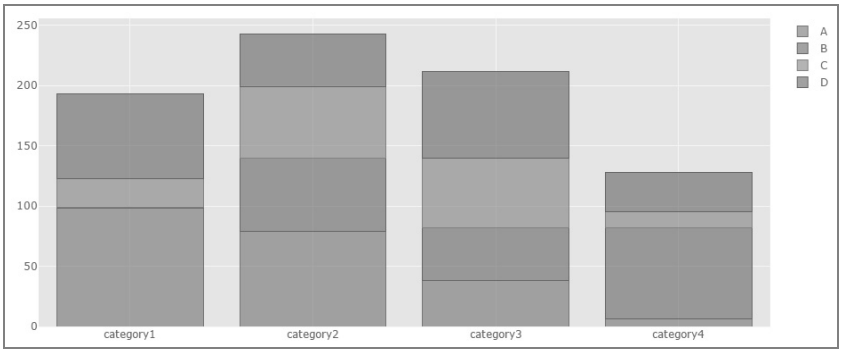

图 4-9　层叠柱状图

3. 散点图

绘制散点图的代码如下。

```
    import plotly.plotly as py
    import cufflinks as cf

    cf.set_config_file(offline=True, world_readable=True)
    df = cf.datagen.lines(2, columns=['中信证券','招商证券'])
    df.iplot(kind='scatter',mode='markers',size=10,symbol='x',colorscale=
'paired',
                          xTitle='日期',yTitle='每股盈利增长率',title='散点图')
```

绘图结果如图 4-10 所示。

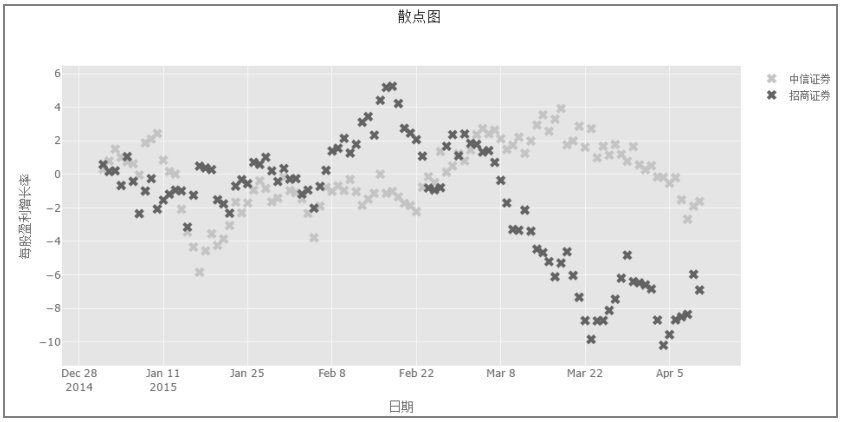

图 4-10　散点图

4．价差图

绘制价差图的代码如下。

```
import cufflinks as cf

cf.set_config_file(offline=True, world_readable=True)
df= cf.datagen.lines(2,columns=['万科A','中粮地产'])
df.iplot(kind='spread',xTitle='日期',yTitle='收益率',title='价差图')
```

绘图结果如图 4-11 所示。

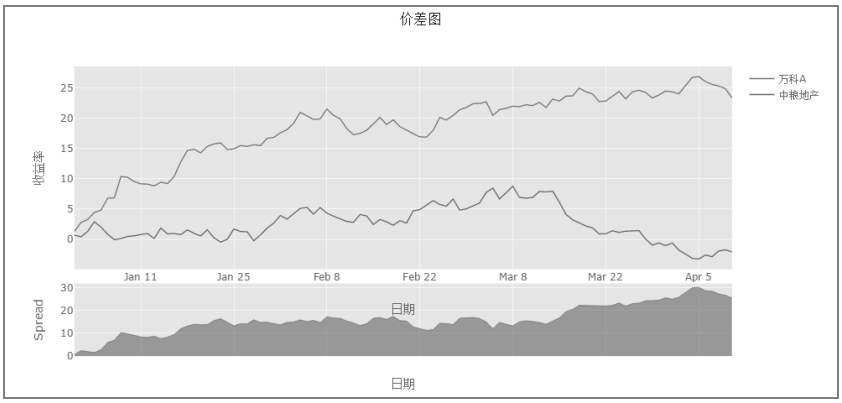

图 4-11　价差图

5. 直方图

绘制直方图的代码如下。

```
import cufflinks as cf

cf.set_config_file(offline=True, world_readable=True)

df= cf.datagen.histogram(2)
df.iplot(kind='histogram',opacity=.75,title='直方图')
```

绘制结果如图 4-12 所示。

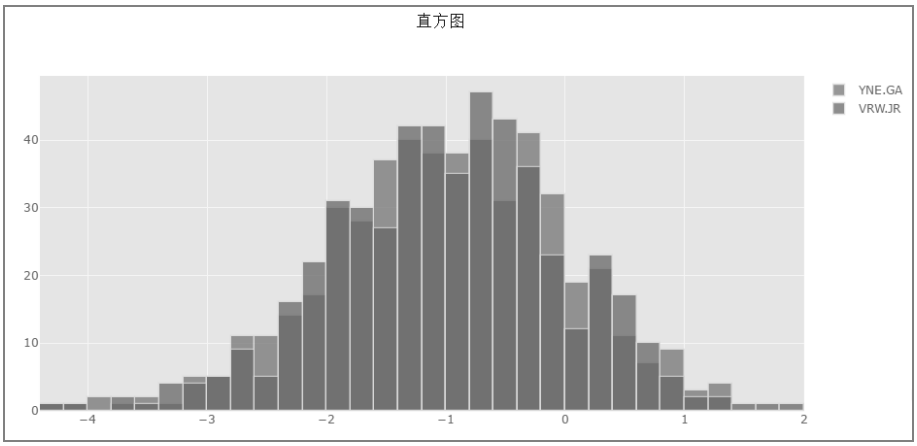

图 4-12　直方图

6. 时间序列子图

绘制时间序列子图的代码如下。

```
import cufflinks as cf

cf.set_config_file(offline=True, world_readable=True, theme='pearl')

df=cf.datagen.lines(4)

df.iplot(subplots=True, shape=(4,1), shared_xaxes=True, fill=True)
```

绘图结果如图 4-13 所示。

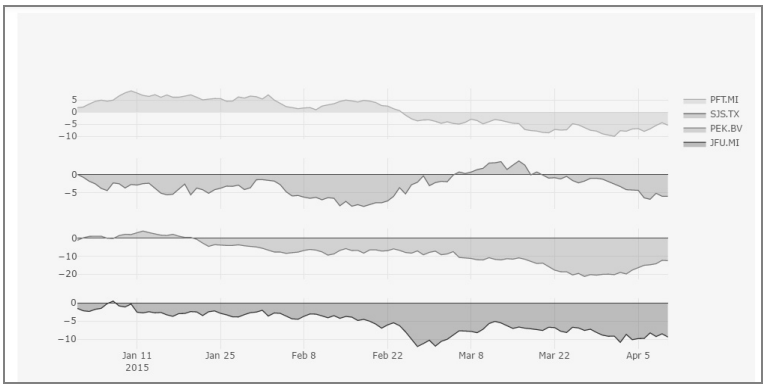

图 4-13　时间序列子图

7．3D 散点图

绘制 3D 散点图的代码如下。

```
    import cufflinks as cf

    cf.set_config_file(offline=True, world_readable=True)

    df = cf.datagen.scatter3d(2,150)
    df.iplot(kind='scatter3d',x='x',y='y',z='z',size=15,categories='categ
ories',
    text='text',title='3D 散点图
',colors=['blue','pink'],width=0.5,margin=(0,0,0,0),opacity=1)
```

绘图结果如图 4-14 所示。

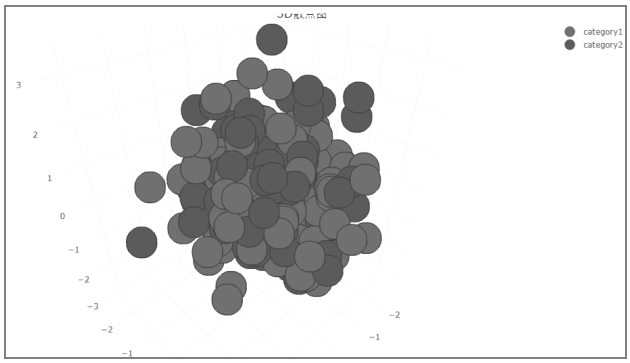

图 4-14　3D 散点图

4.2.6　更多案例

cufflinks 是一个非常简单且易用的模块，读者若想对这个模块有更多的了解，可以参考下面的内容。

（1）想要获取 cufflinks 的源代码或最新版本的信息，可以在下面的网址中找到：https://github.com/santosjorge/cufflinks。

（2）想要获取 cufflinks 的更多案例代码，可以在上面打开的网页中选择 Chart Gallery 链接，也可以在上面打开的网页中下载文件 Cufflinks Tutorial - Chart Gallery.ipynb 并打开，在本地计算机中查看。

（3）想要获取 cufflinks 高级一些的案例代码，可以在上面打开的网页中选择 Pandas Like Visualization 链接，也可以在上面打开的网页中下载文件 Cufflinks Tutorial - Pandas Like.ipynb 并打开，在本地计算机中查看。

第 5 章

金融绘图

K 线图在金融绘图领域中有着重要的地位，而 Plotly 在金融绘图领域有着自己独特的优势。本章主要讲解如何把 Plotly 更好地应用在金融绘图领域中。

5.1 快速绘制K线图

5.1.1 检查 Plotly 版本

导入 Plotly 绘图模块库后检查一下版本是否为最新，截至 2017 年 11 月 17 日，Plotly 最新版本为 2.2.1。

检查 Plotly 版本的代码如下。

```
import plotly
plotly.__version__

'2.2.1'
```

5.1.2 快速绘制 OHLC（美国线）图

Pandas 是金融领域中非常好用且高效的数据分析与处理模块，借助 Pandas 处理

数据的便利性，绘制 K 线图简直不费吹灰之力。

快速绘制 OHLC 图（见文件 first_ohlc.py）的代码如下。

```
    import plotly as py  # 导入 Plotly 库并命名为 py
    import plotly.graph_objs as go

    # -------------pre def
    pyplt = py.offline.plot
    import pandas as pd

    df = pd.read_csv(r'dat/appl.csv', index_col=['date'],
parse_dates=['date'])
    trace = go.Ohlc(
        x=df.index,
        open=df.open,
        high=df.high,
        low=df.low,
        close=df.close,
    )

    data = [trace]

    pyplt(data, filename=r'tmp/first_ohlc.html')
```

在上面的代码中，最核心的是下面的函数。这个函数看起来很简单，基本没有介绍的必要。

```
    import plotly.graph_objs as go

    trace = go.Ohlc(
        x=df.index,
        open=df.open,
        high=df.high,
        low=df.low,
        close=df.close,
    )
```

绘图结果如图 5-1 所示。

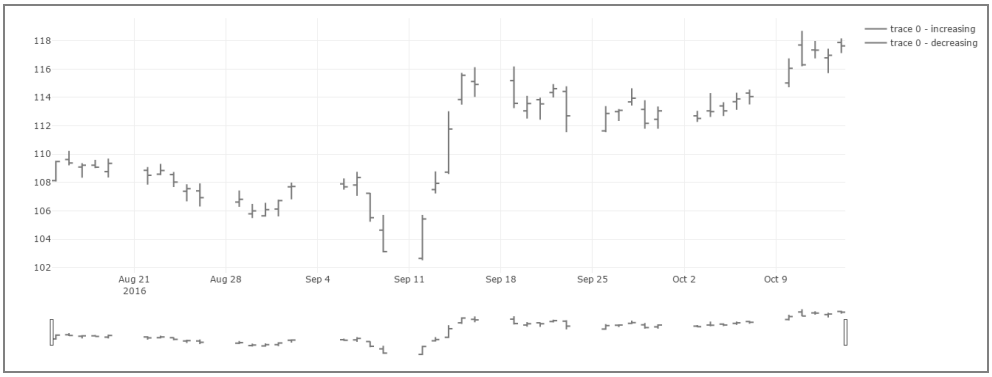

图 5-1　绘制 OHLC 图

从图 5-1 可以看出，go.Ohlc 函数绘制的 K 线图效果非常不错，而且还在 K 线图下面封装了滑块，方便切换时间。go.Ohlc 函数是 Plotly 为了方便用户使用，专门为绘制 K 线图封装的一个对象。笔者之前学习 Plotly 时，绘制 OHLC 图使用的是如下代码（见文件 first_ohlc_old.py）。

```
    import plotly as py  # 导入 Plotly 库并命名为 py
    import plotly.figure_factory  as ff  # 导入 Plotly 工具箱库中的图像工厂方法并命名为 ff

    # -------------pre def
    pyplt = py.offline.plot
    import pandas as pd


    df = pd.read_csv(r'dat/appl.csv', index_col=['date'], parse_dates=['date'])

    fig = ff.create_ohlc(df.open, df.high, df.low, df.close, dates=df.index)
    pyplt(fig, filename=r'tmp/first_ohlc_old.html')
```

绘图结果如图 5-2 所示。

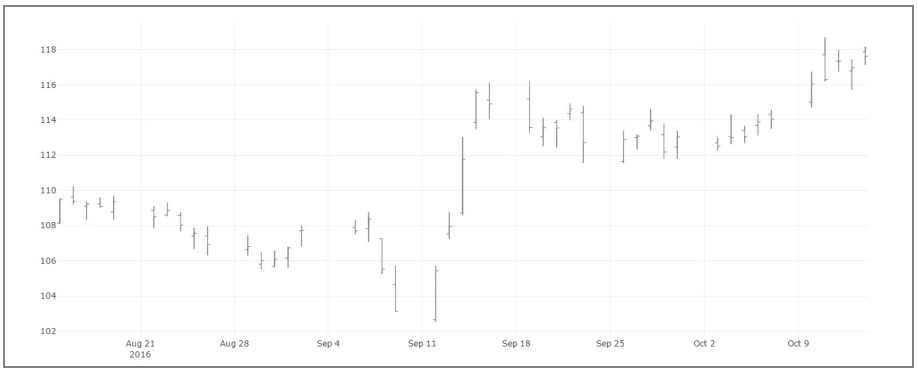

图 5-2　没有滑块的 OHLC 图

可见，除了没有滑块，绘图结果是一样的。

5.1.3　快速绘制蜡烛图

下面编写代码绘制蜡烛图（见文件 first_candlestick.py），代码如下。

```
    import plotly as py  # 导入 Plotly 库并命名为 py
    import plotly.graph_objs as go

    # -------------pre def
    pyplt = py.offline.plot
    import pandas as pd


    df = pd.read_csv(r'dat/appl.csv', index_col=['date'], parse_dates=
['date'])


    trace = go.Candlestick(
        x=df.index,
        open=df.open,
        high=df.high,
        low=df.low,
        close=df.close,
```

```
    )

    data = [trace]

    pyplt(data, filename=r'tmp/first_candlestick.html')
```

以上代码只是把“Ohlc”换成“Candlestick”而已，绘图结果如图 5-3 所示。

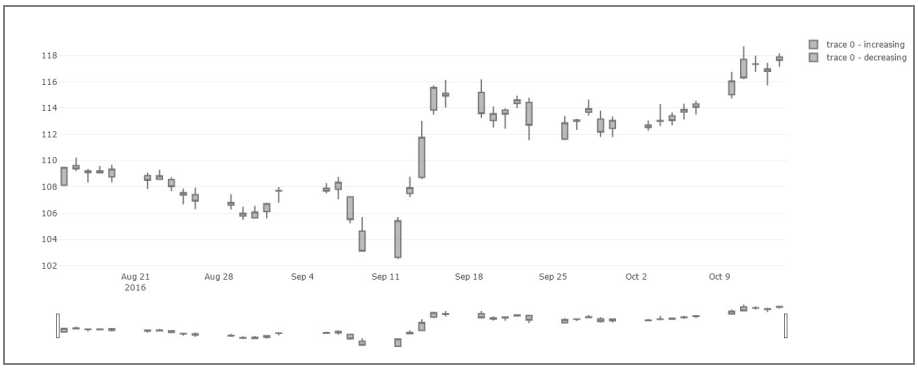

图 5-3　绘制蜡烛图

同样的，给出以前版本绘制蜡烛图的代码（见文件 first_candlestick_old.py）。

```
    import plotly as py  # 导入 Plotly 库并命名为 py
    import plotly.figure_factory  as ff  # 导入 Plotly 工具箱库中的图像工厂方法并命名为 ff



    # -------------pre def
    pyplt = py.offline.plot
    import pandas as pd



    df = pd.read_csv(r'dat/appl.csv', index_col=['date'], parse_dates=['date'])

    fig = ff.create_candlestick(df.open, df.high, df.low, df.close, dates=df.index)

    pyplt(fig, filename=r'tmp/first_candlestick_old.html')
```

绘图结果如图 5-4 所示。

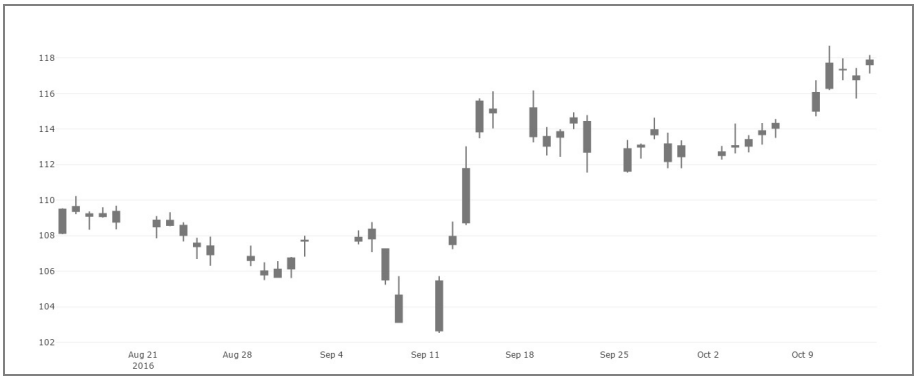

图 5-4　以前版本绘制的蜡烛图

注意

go.Ohlc 函数与 go.Candlestick 函数使用起来虽然简单明了，效果也不错，但是有一个致命的缺陷，那就是绘图结果中包含了所有的日期信息（例如非交易日，图中的 9 月 9 日—9 月 12 日），这个问题会放到下一节去解决。

5.2　K线图的优化

5.2.1　过滤非交易时间

过滤非交易时间属于对 *X* 轴的操作，这个工作需要在布局管理中进行。我们通过 go.Ohlc 函数返回的对象本质上是一个 data，需要对这个 data 添加布局，形成一个完整的 fig（见文件 ohlc_filter_time.py），代码如下。

```
import plotly as py  # 导入 Plotly 库并命名为 py
import plotly.graph_objs as go
import pandas as pd

fig = go.Figure()
```

```
    # -------------pre def
    pyplt = py.offline.plot

    df = pd.read_csv(r'dat/appl.csv', index_col=['date'], parse_dates=
['date'])

    trace = go.Ohlc(
        x=df.index,
        open=df.open,
        high=df.high,
        low=df.low,
        close=df.close,
    )

    fig['data'] = [trace]

    fig['layout'].update(
        xaxis=go.XAxis(
            autorange=True,
            mirror='all',
            gridcolor='rgb(180, 180, 180)',
            showline=True,  # 画出 X 轴
            showgrid=True,
            tickangle=-60,
            categoryorder="category ascending",
            type='category'),
        yaxis=go.YAxis(
            autorange=True,
            gridcolor='rgb(180, 180, 180)',
        ),
    )

    pyplt(fig, filename=r'tmp/ohlc_filter_time.html')
```

绘图结果如图 5-5 所示。

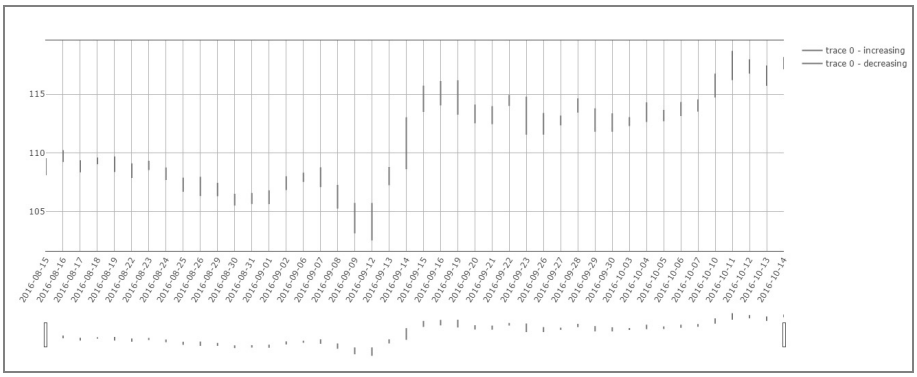

图 5-5　过滤非交易时间的 K 线

在上面的代码中，最关键的是如下两行代码：

```
    categoryorder="category ascending",
    type='category'),
```

这两行代码的意思是，把 *X* 轴的元素看成分类的元素（type='category'），并对其进行升序排序（categoryorder="category ascending"）。既然元素的属性为类别，不是时间，也就没有过滤非交易时间的问题了。

对于 categoryorder 参数，默认是“trace”，即以默认的 df.index 进行排序。它可以是升序（category ascending），也可以是降序（category descending），甚至可以自定义顺序（array），不过这时需要额外的参数支持（categoryarray），用来传递自定义的 index。

对于蜡烛线来说，其过滤方法也是一样的，代码如下（见文件 candlestick_filter_time.py）。

```
import plotly as py  # 导入 Plotly 库并命名为 py
import plotly.graph_objs as go

fig = go.Figure()

# -------------pre def
pyplt = py.offline.plot
import pandas as pd

df = pd.read_csv(r'dat/appl.csv', index_col=['date'], parse_dates=
```

```
['date'])

    trace = go.Candlestick(
        x=df.index,
        open=df.open,
        high=df.high,
        low=df.low,
        close=df.close,
    )

    fig['data'] = [trace]

    fig['layout'] = dict(
        xaxis=dict(
            showline=True, # 画出 X 轴
            tickangle=-60,

            categoryorder="category ascending",
            type='category')
    )


    pyplt(fig, filename=r'tmp/candlestick_filter_time.html')
```

绘图结果如图 5-6 所示。

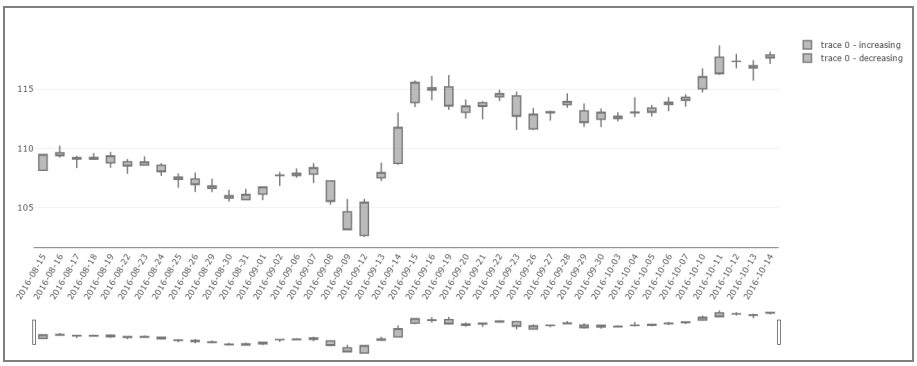

图 5-6　过滤非交易时间的蜡烛线

5.2.2　设置形状、颜色和注释

设置 K 线图的标题、形状、颜色和注释并没有什么特别之处，代码如下（见文件 candlestick_style.py）。

```
    import plotly as py
    import plotly.graph_objs as go
    import pandas as pd

    fig = go.Figure()

    # -------------pre def
    pyplt = py.offline.plot

    df = pd.read_csv(r'dat/appl.csv', index_col=['date'], parse_dates=
['date'])

    trace = go.Candlestick(x=df.index,
                  open=df.open,
                  high=df.high,
                  low=df.low,
                  close=df.close,
                  increasing=dict(line=dict(color='#FF0000')),
                  decreasing=dict(line=dict(color='#0C05F9'))
                  )
    data = [trace]

    fig['data'] = [trace]

    fig['layout'] = {
        'xaxis': {
            'showline': True,  # 画出 X 轴
            'tickangle': -60,
            'categoryorder': "category ascending",
            'type': 'category'},
        'title': '苹果公司 K 线图',
        'yaxis': {'title': '股票价格'},
```

```
    'shapes': [{
        'x0': '2016-08-22', 'x1': '2016-10-05',
        'y0': 0, 'y1': 1, 'xref': 'x', 'yref': 'paper',
        'line': {'color': 'rgb(30,30,30)', 'width': 2}
    }],
    'annotations': [{
        'x': '2016-09-12', 'y': 0.05, 'xref': 'x', 'yref': 'paper',
        'showarrow': True, 'xanchor': 'left',
        'text': '区间最低价'
    }]
}

pyplt(fig, filename=r'tmp/candlestick_style.html')
```

绘图结果如图 5-7 所示。

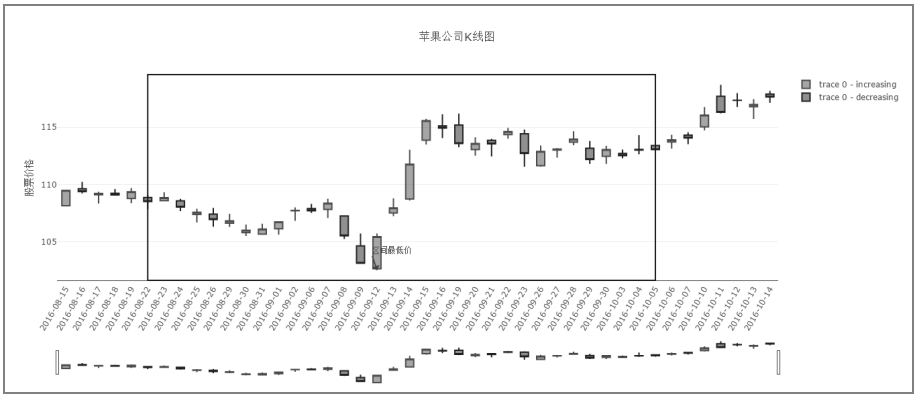

图 5-7　绘图结果

对于颜色，将价格上涨设置为红色，将价格下跌设置为蓝色，这样的设置更符合我国用户的习惯，代码如下。

```
increasing=dict(line=dict(color='#FF0000')),
decreasing=dict(line=dict(color='#0C05F9'))
```

对于标题，可以在 layout 中进行设置，代码如下。

```
'title': '苹果公司 K 线图',
'yaxis': {'title': '股票价格'},
```

设置形状与注释的代码如下。

```
    'shapes': [{
        'x0': '2016-08-22', 'x1': '2016-10-05',
        'y0': 0, 'y1': 1, 'xref': 'x', 'yref': 'paper',
        'line': {'color': 'rgb(30,30,30)', 'width': 2}
    }],
    'annotations': [{
        'x': '2016-09-12', 'y': 0.05, 'xref': 'x', 'yref': 'paper',
        'showarrow': True, 'xanchor': 'left',
        'text': '区间最低价'
    }]
```

读者需要理解 xref 与 yref 这两个参数，这两个参数在本书 3.7 节“绘制 SVG”中介绍过。为了方便读者更好地理解，这里再次说明一下。

xref（yref）参数有 2 个值：paper 和 x（y）。如果值为 x，则 x0、x1 对应的值为坐标轴的绝对值，比如“x0='2016-08-22'，x1='2016-10-05'”表示 shapes（layout 的一个参数，见代码 ohlc_style.py）的 *X* 轴起点在 x0='2016-08-22'处、终点在 x1='2016-10-05'处；如果值为 paper，则 x0、x1 对应的值为坐标轴的相对值。以本案例来说，“y0=0，y1=1”表示 shapes 的 *Y* 轴起点在 y0=0 处、终点在 y1=1 处。

对于 OHLC 图，其绘制方法是一样的（见文件 ohlc_style.py），代码如下。

```
    import plotly as py
    import plotly.graph_objs as go
    import pandas as pd

    fig = go.Figure()

    # -------------pre def
    pyplt = py.offline.plot

    df = pd.read_csv(r'dat/appl.csv', index_col=['date'], parse_dates=
['date'])

    trace = go.Ohlc(x=df.index,
                    open=df.open,
                    high=df.high,
```

```
                low=df.low,
                close=df.close,
                increasing=dict(line=dict(color='#FF0000')),
                decreasing=dict(line=dict(color='#0C05F9'))
                )
data = [trace]

fig['data'] = [trace]

fig['layout'] = {
    'xaxis': {
        'showline': True,  # 画出 X 轴
        'tickangle': -60,
        'categoryorder': "category ascending",
        'type': 'category'},
    'title': '苹果公司 K 线图',
    'yaxis': {'title': '股票价格'},
    'shapes': [{
        'x0': '2016-08-22', 'x1': '2016-10-05',
        'y0': 0, 'y1': 1, 'xref': 'x', 'yref': 'paper',
        'line': {'color': 'rgb(30,30,30)', 'width': 2}
    }],
    'annotations': [{
        'x': '2016-09-12', 'y': 0.05, 'xref': 'x', 'yref': 'paper',
        'showarrow': True, 'xanchor': 'left',
        'text': '区间最低价'
    }]
}

pyplt(fig, filename=r'tmp/ohlc_style.html')
```

绘图结果如图 5-8 所示。

注意

这里绘制的 OHLC 图没有开盘价与收盘价，这应该是 Plotly 的 bug，希望在下一个版本中 Plotly 可以解决这个问题。

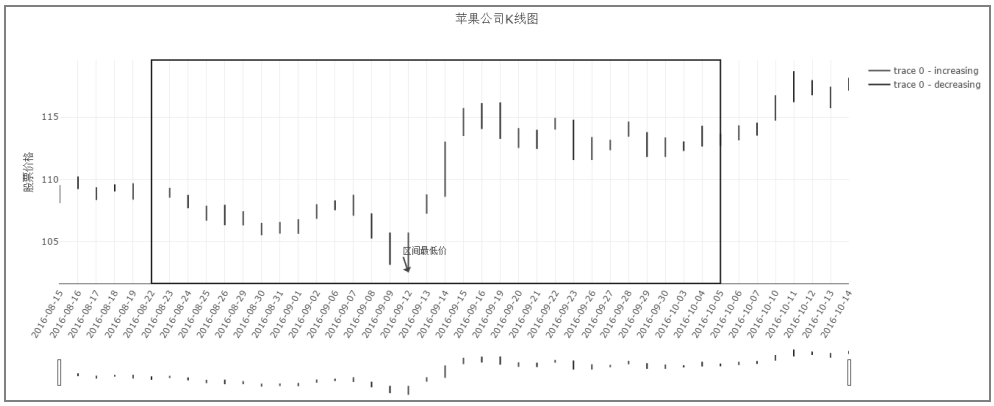

图 5-8　OHLC 图绘图结果

5.2.3　添加技术指标

在实际工作中，K 线图往往是需要结合其他技术指标进行判断的，因此我们需要绘制一个复合的 K 线图。本节主要介绍 K 线图与 5 日均线结合的方法，对于其他的技术指标如 MACD、RSI、KDJ 等，可以根据这个方法进行扩展，代码如下（见文件 candlestick_add_avg.py）。

```
    import plotly as py  # 导入 Plotly 库并命名为 py
    import plotly.graph_objs as go
    import pandas as pd

    fig = go.Figure()

    # -------------pre def
    pyplt = py.offline.plot

    df = pd.read_csv(r'dat/appl.csv', index_col=['date'], parse_dates=
['date'])
    df['avg_5'] = df['close'].rolling(5).mean()

    # K 线图
    trace = go.Candlestick(
```

```
    x=df.index,
    open=df.open,
    high=df.high,
    low=df.low,
    close=df.close,
)


# 5 日均线拟合直线
add_line = go.Scatter(
        x=df.index,
        y=df.avg_5,
        name= '5 日均线',
        line=dict(color='black')
    )

fig['data'] = [trace]
fig['data'].extend([add_line])

fig['layout'].update(
    xaxis=go.XAxis(
        autorange=True,
        mirror='all',
        gridcolor='rgb(180, 180, 180)',
        showline=True,  # 画出 X 轴
        showgrid=True,
        tickangle=-60,
        categoryorder="category ascending",
        type='category'),
    yaxis=go.YAxis(
        autorange=True,
        gridcolor='rgb(180, 180, 180)',
    ),
)

pyplt(fig, filename=r'tmp/candlestick_add_svg.html')
```

绘图结果如图 5-9 所示。

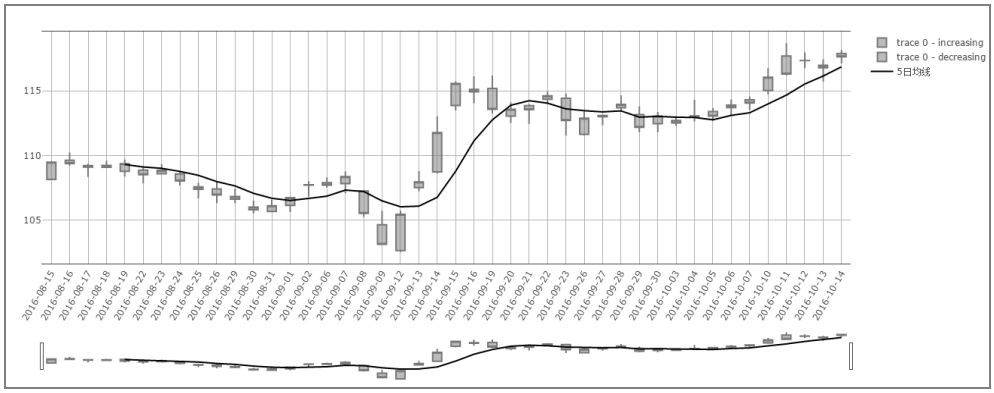

图 5-9　K 线图与 5 日均线结合

本案例的要点如下。

```
# 5 日均线拟合直线
add_line = go.Scatter(
        x=df.index,
        y=df.avg_5,
        name= '5 日均线',
        line=dict(color='black')
    )

fig['data'] = [trace]
fig['data'].extend([add_line])
```

可以看出，绘制方法与之前所学的知识一样，这里不再赘述。

对于 OHLC 图，绘制方法相同（见文件 ohlc_add_avg.py），代码如下。

```
import plotly as py  # 导入 Plotly 库并命名为 py
import plotly.graph_objs as go
import pandas as pd

fig = go.Figure()

# -------------pre def
pyplt = py.offline.plot
```

```
    df = pd.read_csv(r'dat/appl.csv', index_col=['date'], parse_dates=
['date'])

    # K 线图
    trace = go.Ohlc(
        x=df.index,
        open=df.open,
        high=df.high,
        low=df.low,
        close=df.close,
    )


    # 5 日均线拟合直线
    add_line = go.Scatter(
            x=df.index,
            y=df.open,
            name= 'Open Vals',
            line=dict(color='black')
        )

    fig['data'] = [trace]
    fig['data'].extend([add_line])

    fig['layout'].update(
        xaxis=go.XAxis(
            autorange=True,
            mirror='all',
            gridcolor='rgb(180, 180, 180)',
            showline=True,  # 画出 X 轴
            showgrid=True,
            tickangle=-60,
            categoryorder="category ascending",
            type='category'),
        yaxis=go.YAxis(
            autorange=True,
```

```
        gridcolor='rgb(180, 180, 180)',
    ),
)

pyplt(fig, filename=r'tmp/ohlc_add_svg.html')
```

绘图结果如图 5-10 所示。

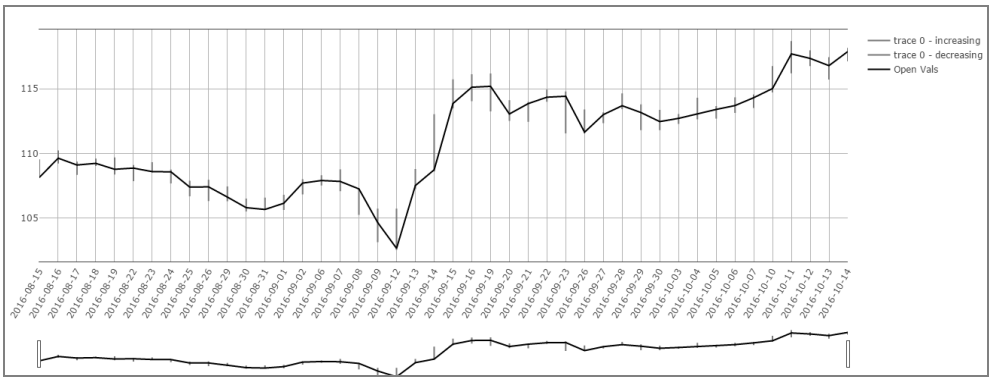

图 5-10　OHLC 绘制结果

注意

与之前的情况一样，这里绘制的 OHLC 线仍然没有开盘价与收盘价，这应该是 Plotly 的 bug，希望在下一个版本中 Plotly 可以解决这个问题。

5.3　使用自定义数据的金融绘图

创建 K 线图不一定要使用标准的 OHLC 价格图，可以使用自定义的数据，方法非常简单（见文件 ohlc_custom_data.py），代码如下。

```
import plotly as py
import plotly.graph_objs as go
from datetime import datetime


# -------------pre def
pyplt = py.offline.plot
```

```
# 添加数据
open_data = [33.0, 33.3, 33.5, 33.0, 34.1]
high_data = [33.1, 33.3, 33.6, 33.2, 34.8]
low_data = [32.7, 32.7, 32.8, 32.6, 32.8]
close_data = [33.0, 32.9, 33.3, 33.1, 33.1]
dates = [datetime(year=2016, month=10, day=10),
        datetime(year=2016, month=11, day=10),
        datetime(year=2016, month=12, day=10),
        datetime(year=2017, month=1, day=10),
        datetime(year=2017, month=2, day=10)]


# 创建 OHLC
trace = go.Ohlc(x=dates,
              open=open_data,
              high=high_data,
              low=low_data,
              close=close_data)

data = [trace]

pyplt(data, filename=r'tmp/ohlc_custom_data.html')
```

绘制结果如图 5-11 所示。

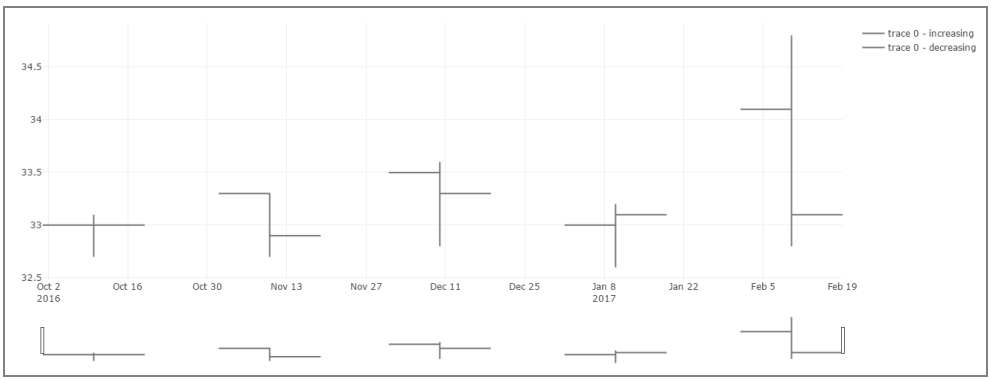

图 5-11　使用自定义数据绘制价格图

对于蜡烛图，代码如下（见文件 candlestick_custom_data.py）。

```
import plotly as py
import plotly.graph_objs as go
from datetime import datetime


# -------------pre def
pyplt = py.offline.plot

# 添加数据
open_data = [33.0, 33.3, 33.5, 33.0, 34.1]
high_data = [33.1, 33.3, 33.6, 33.2, 34.8]
low_data = [32.7, 32.7, 32.8, 32.6, 32.8]
close_data = [33.0, 32.9, 33.3, 33.1, 33.1]
dates = [datetime(year=2016, month=10, day=10),
        datetime(year=2016, month=11, day=10),
        datetime(year=2016, month=12, day=10),
        datetime(year=2017, month=1, day=10),
        datetime(year=2017, month=2, day=10)]


# 创建 OHLC
trace = go.Candlestick(x=dates,
              open=open_data,
              high=high_data,
              low=low_data,
              close=close_data)

data = [trace]

pyplt(data, filename=r'tmp/candlestick_custom_data.html')
```

绘图结果如图 5-12 所示。

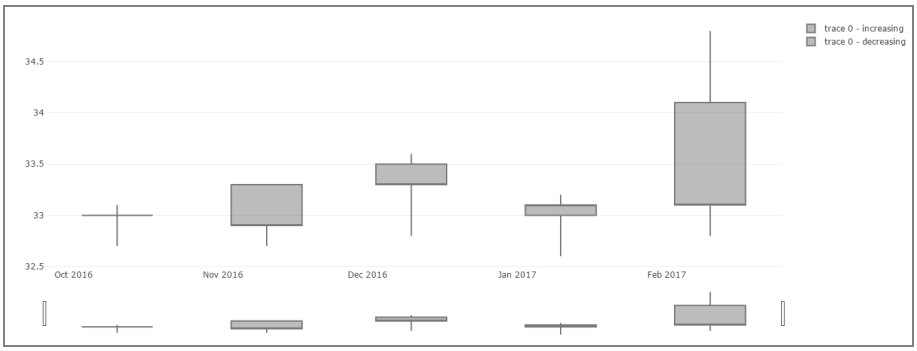

图 5-12　使用自定义数据绘制蜡烛图

5.4　高级金融绘图

前面讲解了手工添加技术指标的方法，这种方法比较笨，虽然可以利用 talib 等技术指标的专业模块来搞定这些问题，可是对于新手来说还是比较困难的。幸运的是，cufflinks 开发团队已经帮我们解决了这个问题：从 cufflinks 的 v0.11 版本开始，增加了 QuantFig 函数，使用这个函数能让金融绘图更加方便。

5.4.1　入门案例

QuantFig 函数的使用非常简单，与一般 cufflinks 函数的使用方法相比，只加了一行代码，见文件“高级金融绘图.ipynb（ 5.4.1 入门案例）”，代码如下。

```
import cufflinks as cf

cf.set_config_file(offline=True, world_readable=True)

df=cf.datagen.ohlc()
qf=cf.QuantFig(df,title='高级金融绘图',legend='top',name='QF')
qf.add_bollinger_bands()
qf.iplot()
```

绘图结果如图 5-13 所示。

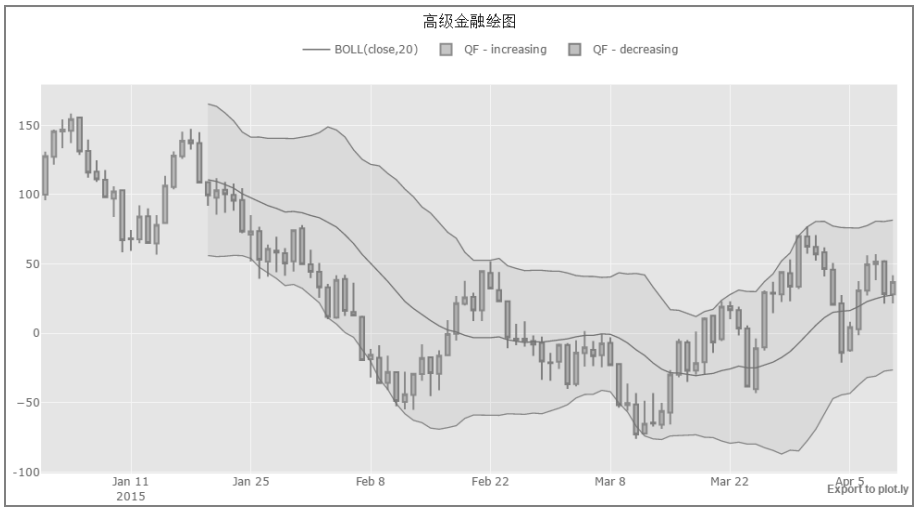

图 5-13　高级金融绘图结果

从绘图结果可以看出，我们以很简单的方式绘制了 K 线图并在 K 线图上添加了布林线。

这个案例的关键代码是下面这一行：

```
qf=cf.QuantFig(df,title='高级金融绘图',legend='top',name='QF')
```

这行代码的作用是把传统的 DataFrame 对象转换成 QuantFig 对象，从而方便我们绘图。

5.4.2　综合案例

在实际工作中，要添加更多的指标也非常简单，见文件“高级金融绘图.ipynb（5.4.2 综合案例）”，代码如下。

```
import cufflinks as cf


cf.set_config_file(offline=True, world_readable=True)

df=cf.datagen.ohlcv()
```

```
    qf=cf.QuantFig(df,title='高级金融绘图',legend='top',name='QF')
    qf.add_bollinger_bands()
    qf.add_sma([10,20],width=2,color=['green','lightgreen'],legendgroup=T
rue)
    qf.add_rsi(periods=20,color='java')
    qf.add_bollinger_bands(periods=20,boll_std=2,colors=['magenta','grey'
],fill=True)
    qf.add_volume()
    qf.add_macd()
    qf.iplot()
```

绘图结果如图 5-14 所示。

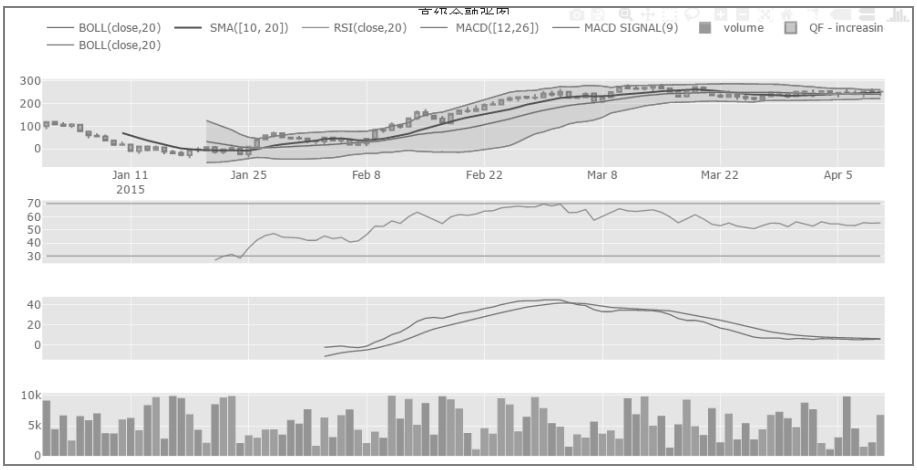

图 5-14　添加更多指标的高级金融绘图结果

由此可见，我们通过简单的几行代码就创建了 K 线图的多种技术指标，如 MACD、RSI、布林线等。这些指标都在 QuantFig 中封装好了，我们只需要调用就可以了。QuantFig 函数还能自动添加子图，总体效果看起来非常不错。

第 6 章

Matplotlib

6.1 Matplotlib简介

Matplotlib 是 Python 的一套基于 NumPy 的绘图工具包，并且是 Python 中最著名的绘图包之一。Matplotlib 提供了一整套和 MATLAB 相似的命令 API，十分适合交互式绘制图表，并且 Matplotlib 可以方便地将 API 作为绘图控件，嵌入到 GUI 应用程序中。

Matplotlib 功能完善，其风格跟 MATLAB 相似，同时也继承了 Python 简单明了的语法风格，可以很方便地设计和输出二维和三维的图表，对于常见的坐标系，如笛卡儿坐标系、极坐标系、球坐标系、三维坐标系等都能够很好地支持。

虽然 Matplotlib 最初是在模仿 MATLAB 图形命令时被设计的，但是它的底层使用的却是 Python 语法风格和面向对象的方式，与 MATLAB 没有太大的关系，两者是相互独立的。为了提高处理大量数据的性能，Matplotlib 大量使用了 NumPy 和其相关的扩展代码。为了方便快速绘图，Matplotlib 通过 pyplot 模块提供了一套和 MATLAB 类似的绘图 API，我们只需要调用 pyplot 模块所提供的函数，就可以实现快速绘图及设置图表的各种细节。

为了将面向对象的绘图库包装成只使用函数的调用接口，pyplot 模块的内部保存了当前图表及当前子图等信息。

另外，pylab 是 Matplotlib 面向对象绘图库的另一个接口。pylab 的语法和 MATLAB 十分相近，主要的绘图命令和 MATLAB 对应的命令也有相似的参数。pylab 接口由 matplotlib.pylab 提供函数集，允许用户使用非常类似于 MATLAB 绘图的方式快速创建绘图。

由于 Matplotlib 功能强大，所以受到绘图工作者和科研工作者的青睐。下面从绘图的第一步：安装 Matplotlib 开始介绍。

6.2 安装Matplotlib

想要绘图，就需要下载并安装 Matplotlib。笔者推荐用 pip 命令来安装，进入 CMD 窗口中，通过 pip install matplotlib 命令进行安装，系统会自动下载安装包，如图 6-1 所示。

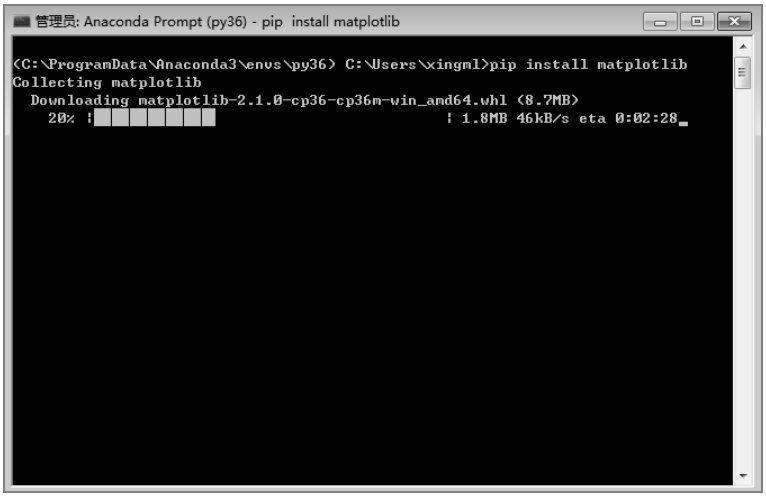

图 6-1 系统自动下载安装包

正常情况下，系统还会下载其他的关联安装包并完成安装，最后系统会提示用户 Matplotlib 已经安装成功，如图 6-2 所示。

进入 Python IDLE 中，运行“import matplotlib”，如图 6-3 所示，就可以开始使用 Matplotlib 了。如果不想自己安装 Matplotlib，可以下载 Anaconda 等 Python 的集成工具。Anaconda 是一个开源的 Python 发行版本，包含了 Conda、Python 等 180

多个科学包及其依赖项。因为包含了大量的科学包，所以 Anaconda 的下载文件比较大（约 500 MB）。Anaconda 提供了很多用于科学计算的模块，常见的包括 NumPy、SciPy 和 Matplotlib。Anaconda 官方下载地址为 https://www.anaconda.com/download/，选择相应的版本下载并安装即可。

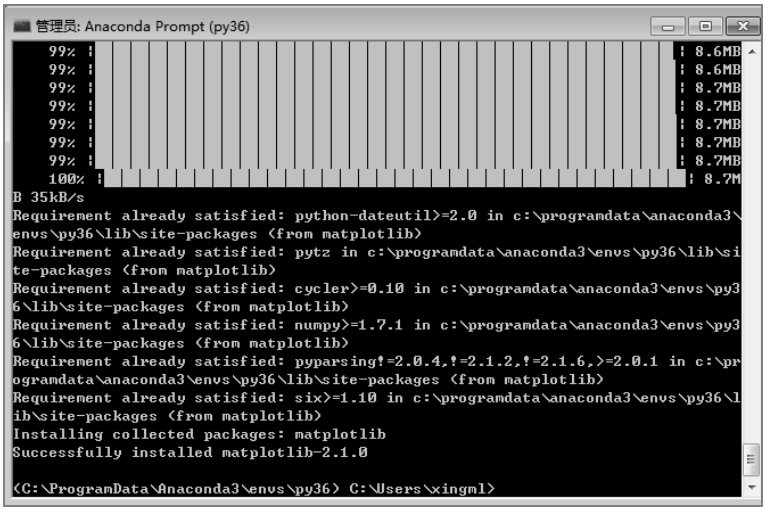

图 6-2　下载关联安装包并提示安装成功

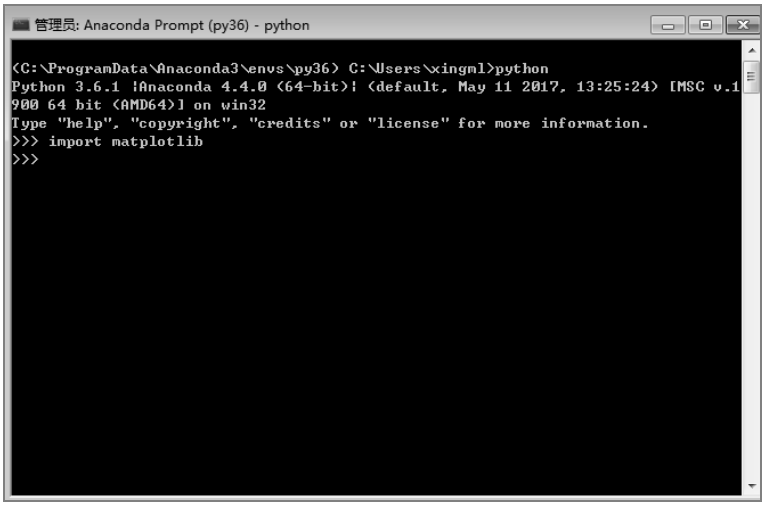

图 6-3　运行“import matplotlib”

6.3 调整Matplotlib参数

安装好 Matplotlib 后，要为绘图项目设置 Matplotlib 参数。

在代码执行的过程中，有以下两种方式可以调整参数。

1. 使用参数字典

使用参数字典调整参数的代码如下。

```
import matplotlib as plt
#导入模块包
print(plt.rc_params())
```

打印参数，由于参数众多，所以我们把输出的参数进行简化，得到如下参数。

```
_internal.classic_mode: False
agg.path.chunksize: 0
animation.avconv_args: []
axes.facecolor: w
axes.labelweight: normal
backend: TkAgg
boxplot.whiskers: 1.5
contour.corner_mask: True
contour.negative_linestyle: dashed
datapath: D:\zwPython\py35\python-3.5.3.amd64\lib\site-packages\
matplotlib\mpl-data
figure.autolayout: False
figure.titleweight: normal
font.cursive: ['Apple Chancery', 'Textile', 'Zapf Chancery', 'Sand',
'Script MT', 'Felipa', 'cursive']
font.family: ['sans-serif']
grid.alpha: 1.0
grid.color: #b0b0b0
grid.linestyle: -
grid.linewidth: 0.8
hatch.color: k
hatch.linewidth: 1.0
hist.bins: 10
```

```
image.aspect: equal
keymap.all_axes: ['a']
legend.borderaxespad: 0.5
lines.antialiased: True
markers.fillstyle: full
patch.antialiased: True
savefig.bbox: None
text.color: k
timezone: UTC
tk.window_focus: False
toolbar: toolbar2
verbose.fileo: sys.stdout
xtick.bottom: True
ytick.color: k
```

在以上参数中，常用的参数解释如下。

- axes：设置坐标轴边界、颜色、坐标刻度值大小和网格的显示。
- figure：设置边界颜色、图形大小和子区（subplot）。
- font：设置字体、字号和样式。
- grid：设置网格颜色和线型。
- legend：设置图例和其中的文本的显示。
- lines：设置线条（颜色、线型、宽度等）和标记。
- savefig：对保存的图形进行单独设置。
- xtick 和 ytick：为 *X*、*Y* 轴的主刻度和次刻度设置颜色、大小、方向和标签大小。

接下来，调用 matplotlib.rc()命令，通过传入关键字修改参数，代码如下。

```
import matplotlib as plt
import pylab
pylab.plot([1,2,3,4,5,6,7,8,9,10])
plt.rc("lines", marker="x", linewidth=12, color="red")
pylab.show()
```

2．修改配置文件

可以用 matplot.matplotlib_fname()命令查找当前用户的配置文件目录，从而修改 Matplotlib 的文件参数，代码如下。

```
    import matplotlib as plt
    print(plt.matplotlib_fname())
```

获取配置文件的路径，笔者的配置文件的计算机路径输出为：

```
    D: \python-3.5.3.amd64\lib\site-packages\matplotlib\mpl-data\
matplotlibrc
```

找到上述路径，再找到 matplotlibrc 文件，用编辑器打开，即可修改配置参数，参数截图如图 6-4 所示。

```
### Boxplot
#boxplot.notch       : False
#boxplot.vertical    : True
#boxplot.whiskers    : 1.5
#boxplot.bootstrap   : None
#boxplot.patchartist : False
#boxplot.showmeans   : False
#boxplot.showcaps    : True
#boxplot.showbox     : True
#boxplot.showfliers  : True
#boxplot.meanline    : False

#boxplot.flierprops.color           : 'k'
#boxplot.flierprops.marker          : 'o'
#boxplot.flierprops.markerfacecolor : 'none'
#boxplot.flierprops.markeredgecolor : 'k'
#boxplot.flierprops.markersize      : 6
#boxplot.flierprops.linestyle       : 'none'
#boxplot.flierprops.linewidth       : 1.0

#boxplot.boxprops.color     : 'k'
#boxplot.boxprops.linewidth : 1.0
#boxplot.boxprops.linestyle : '-'

#boxplot.whiskerprops.color     : 'k'
#boxplot.whiskerprops.linewidth : 1.0
#boxplot.whiskerprops.linestyle : '-'
```

图 6-4　配置参数

6.4　常用的API功能

下面列出 matplotlib.pyplot 的一些常用函数，需要注意的是，假设我们已经通过代码 import matplotlib.pyplot as plt 导入了 Matplotlib，并且下面的所有函数都省略了“plt.”，例如我们看到的 figure()函数实际上是 plt.figure()函数。

- figure()：多次使用 figure 命令生成多个图时，图片号按顺序增加。
- text()：添加文字说明，可以在图中的任意位置添加文字，并支持 LaTeX 语法。
- xlable()、ylable()：用于添加 *X* 轴和 *Y* 轴标签。
- title()：用于添加图的标题。

- axis([xmin, xmax, ymin, ymax])：用于确定坐标范围。
- xlim(xmin, xmax)和 ylim(ymin, ymax)：用来调整坐标范围。
- annotate()：文本注释，在数据可视化的过程中，图片中的文字经常被用来注释图中的一些特征。使用 annotate()方法可以很方便地添加此类注释。在使用 annotate()时，要考虑两个点的坐标，即被注释的地方 xy(x, y)和插入文本的地方 xytext(x, y)。
- xticks()、yticks()：设置轴记号，人为设置坐标轴刻度显示的值。
- subplot()：设置子图，例如 subplot(2,3,1)表示把图分割成 2×3 的网格，也可以简写为 subplot(231)。其中，第一个参数是行数，第二个参数是列数，第三个参数表示图形的标号。
- axes()：在 Matplotlib 中，整个图像为一个 Figure 对象。Figure 对象中可以包含一个或多个 axes 对象。每个 axes 对象都是一个拥有自己坐标系统的绘图区域。

6.5　线性函数

Matplotlib 中最基础的模块是 pyplot。下面从最简单的点图和线图开始讲解，比如有一组数据，还有一个拟合模型，通过编写代码来实现数据与模型结果的可视化。

假设一个线性函数具有形式 $y=f=ax+b$，自变量为 x，因变量为 y，Y 轴截距为 b。下面用简单的数据来描述线性方程（见文件 simple_line.py），代码如下。

```
import matplotlib.pyplot as plt
#使用 import 导入模块 matplotlib.pyplot 并简写成 plt
import numpy as np
#使用 import 导入模块 NumPy 并简写成 np
import plotly as py  # 导入 Plotly 库并命名为 py

# -------------pre def
pympl = py.offline.plot_mpl

# 配置中文显示
plt.rcParams['font.family'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
```

```
    fig, ax = plt.subplots()
    x = np.linspace(1,10)
    # np.linspace(start, stop, num=50, endpoint=True, retstep=False,
dtype=None)
    # 参数解释：start 和 stop 是开始、结束的数字，num 是生成多少个数字，默认为 50 个；
endpoint 是最后一个 stop 数字是否包含进去，默认包含；retstep 是两个数字间的间距，默认
不显示；dtype 默认为 None
    y = x*3 + 5
    #线性方程 y= x*3 + 5
    plt.title("线性函数")
    #设置标题
    plt.xlabel("x 值")
    #设置 X 轴标签
    plt.ylabel("y 值")
    #设置 Y 轴标签
    plt.plot(x, y)
    #画图

    plot_url = pympl(fig,filename=r'tmp/simple_line.html', show_link=False,
resize=True)
    #离线显示图片
```

绘图结果如图 6-5 所示。

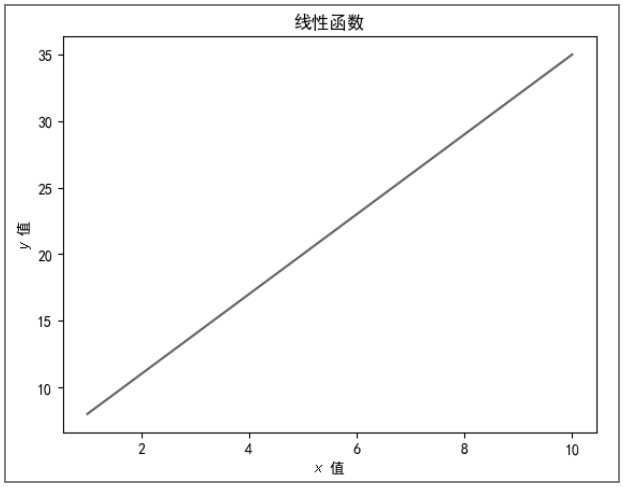

图 6-5　线性函数绘图结果

6.6　增加子图

1．案例1

在绘图时，有时需要把几幅图放在一张图表中。在 Matplotlib 中，一个 Figure 对象可以包含多个子图（axes），可以使用 subplot()函数快速绘制，其调用形式如下：

```
subplot(numRows, numCols, plotNum)
```

整个绘图区域被分成 numRows 行和 numCols 列，plotNum 参数指定创建的 axes 对象所在的区域，如果 numRows＝2、numCols＝2，那么整个绘图区域为 2×2 的图片区域，用坐标表示为（1，1）、（1，2）、（2，1）、（2，2）。本案例见文件 subplot_1.py，代码如下。

```
import numpy as np#使用 import 导入模块 NumPy
import matplotlib.pyplot as plt#使用 import 导入模块 matplotlib.pyplot
import plotly as py  # 导入 Plotly 库并命名为 py

# -------------pre def
pympl = py.offline.plot_mpl

fig, ax = plt.subplots()

# 配置中文显示
plt.rcParams['font.family'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号



x1 = np.arange(0.0, 3.0, 0.01)
x2 = np.arange(0.0, 4.0, 0.01)
#生成 x1、x2 数据
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)
#生成 y1、y2 数据
plt.subplot(2, 1, 1)
# pyplot 通过调用 subplot 或 add_subplot 增加子图，如 p1=plt.subplot(211) 或 p1= plt.subplot(2,1,1)， 表示创建一个 2 行 1 列的图，p1 为第一个子图，然后在 p1 上画曲
```

```
线，设置标注、标题、图例等，接着就可以使用 p1 调用相关的函数了，还可以直接使用 pyplot 画图、添加标签等
    plt.plot(x1, y1)
    plt.title('子图 1')
    #添加标题
    plt.ylabel('y1 值')
    #添加 Y 轴名称
    plt.subplot(2, 1, 2)
    #添加子图
    plt.plot(x2, y2)
    plt.xlabel('数量')
    #添加标签
    plt.ylabel('y2 值')
    #添加 Y 轴名称
    plot_url = pympl(fig,filename=r'tmp/subplot_1.html', show_link=False, resize=True)
    #离线画图
```

绘图结果如图 6-6 所示。

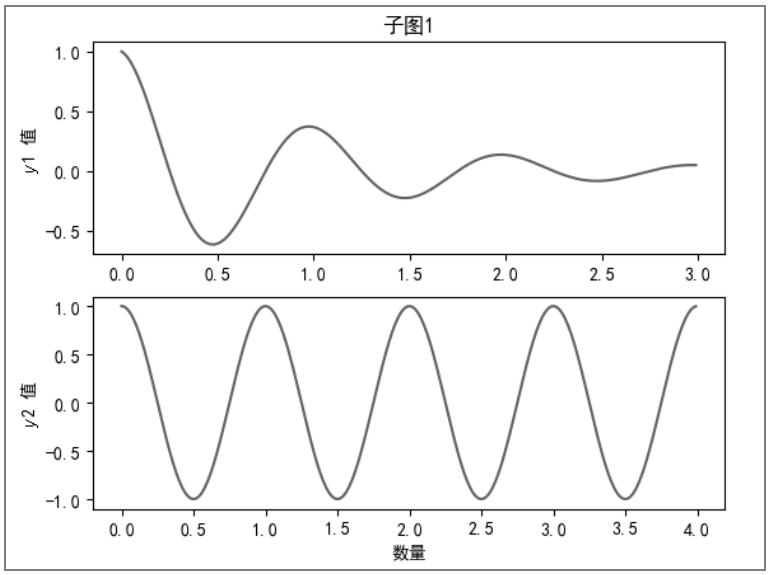

图 6-6　增加子图绘图结果 1

2．案例2

本案例见文件 subplot_2.py，代码如下。

```
    import numpy as np#使用 import 导入模块 NumPy
    import matplotlib.pyplot as plt#使用 import 导入模块 matplotlib.pyplot
    import plotly as py  # 导入 Plotly 库并命名为 py
    # -------------pre def
    pympl = py.offline.plot_mpl
    # 配置中文显示
    fig, ax = plt.subplots()
    plt.rcParams['font.family'] = ['SimHei']  # 用来正常显示中文标签
    plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

    def f(t):
        return np.cos(2*np.pi*t)
    #生成 f(t)函数
    t1 = np.arange(0.0, 5.0, 0.5)
    t2 = np.arange(0.0, 5.0, 0.01)
    #生成 t1、t2 数据
    plt.figure(1)
    plt.subplot(211)
    #添加子图
    plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
    #画图
    plt.subplot(212)
    #添加子图
    plt.plot(np.cos(2*np.pi*t2), 'r--')

    plot_url = pympl(fig,filename=r'tmp/subplot_2.html', show_link=False,
resize=True)
    #离线画图
```

绘图结果如图 6-7 所示。

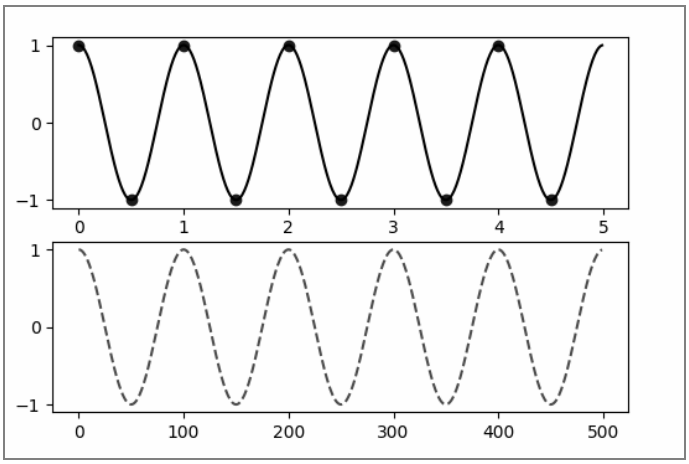

图 6-7　增加子图绘图结果 2

6.7　确定坐标范围

在 Matplotlib 中，使用 xlim(x,y)和 ylim(x,y)函数可以确定坐标范围（见文件 axis_limit.py），代码如下。

```
    from pylab import *
    import numpy as np
    #使用 import 导入模块 NumPy
    import matplotlib.pyplot as plt
    #使用 import 导入模块 matplotlib.pyplot
    import plotly as py
    # 导入 Plotly 库并命名为 py
    # -------------pre def
    pympl = py.offline.plot_mpl

    # 配置中文显示
    plt.rcParams['font.family'] = ['SimHei']  # 用来正常显示中文标签
    plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

    x = np.arange(-5.0, 5.0, 0.02)
    # x = np.linspace(-5, 5, num=50, endpoint=True, retstep=False,
dtype=None)
```

```
    y1 = x**2+1
    plt.figure(1)
    plt.subplot(211)
    xlabel('x 值')
    ylabel('y1 值')
    #设置 X 轴范围，X 轴刻度只显示（-5,3），总区间为（-5,5）
    xlim(-5, 3)
    #设置 Y 轴范围
    ylim(-10, 50)
    plt.plot(x,y1)
    fig= plt.gcf()
    plot_url = pympl(fig,filename=r'tmp/axis_limit.html',
show_link=False,resize=True)
    #离线画图
```

绘图结果如图 6-8 所示。

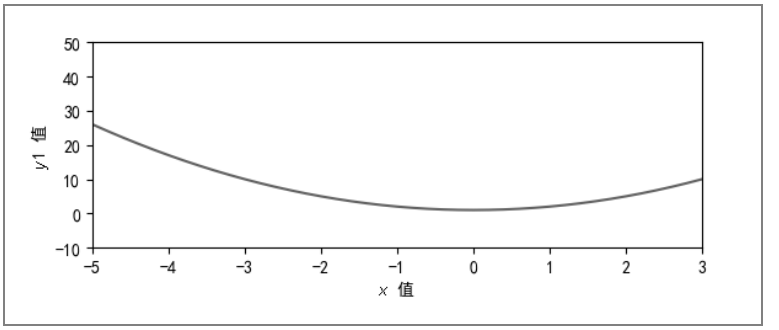

图 6-8　确定坐标范围的绘图

6.8　概率图

概率图模型是图灵奖获得者 Pearl 提出的用图来表示变量之间概率依赖关系的理论。概率图模型的理论分为概率图模型表示理论、概率图模型推理理论和概率图模型学习理论。

正态分布（Normal Distribution），也称为常态分布，又名高斯分布（Gaussian Distribution），是一个在数学、物理及工程等领域都非常重要的概率分布，在统计学

的许多方面有着重大的影响力。正态分布最早由 A.棣莫弗在求二项分布的渐近公式中得到。正态曲线呈钟形，两头低，中间高，左右对称，因其曲线呈钟形，所以人们经常称之为钟形曲线，其形状如图 6-9 所示。

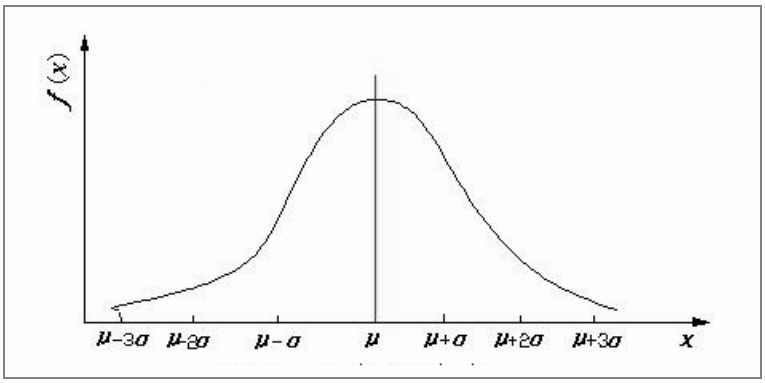

图 6-9　正态分布曲线

在实际应用中，经常使用的是正态概率密度函数 normpdf。normpdf 的语法格式为：normpdf（X,mu,sigma）。其中，X 为一个向量，mu 为均值，sigma 为标准差（本案例见文件 probability_density.py），代码如下。

```
import matplotlib.mlab as mlab
#使用 import 导入模块 matplotlib.mlab
import numpy as np
#使用 import 导入模块 NumPy
import matplotlib.pyplot as plt
#使用 import 导入模块 matplotlib.pyplot
import plotly as py
# 导入 Plotly 库并命名为 py
# -------------pre def
pympl = py.offline.plot_mpl


# 配置中文显示
plt.rcParams['font.family'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
```

```
    #正态分布函数和其他函数都在 matplotlib.mlab 库里面
    np.random.seed(19680801)
    #生成随机种子
    mu = 100
    sigma = 15
    x = mu + sigma * np.random.randn(437)
    #生成数据
    num_bins = 50
    fig, ax = plt.subplots()
    n, bins, patches = ax.hist(x, num_bins, normed=1)
    #生成直方图


    y = mlab.normpdf(bins, mu, sigma)
    #normpdf: 正态概率密度函数
    #Y = normpdf(X,mu,sigma)
    #mu: 均值
    #sigma: 标准差
    #Y: 正态概率密度函数在 x 处的值
    ax.plot(bins, y, '--')
    #画图
    ax.set_xlabel('X 轴')
    #X 轴标签
    ax.set_ylabel('概率密度')
    #Y 轴标签
    ax.set_title(r'柱状图 IQ: $\mu=100$, $\sigma=15$')
    #设置标题
    fig.tight_layout()
    #生成紧凑的图片
    plot_url = pympl(fig,filename=r'tmp/probability_density.html',
show_link=False,resize=True)
    #离线画图
```

绘图结果如图 6-10 所示。

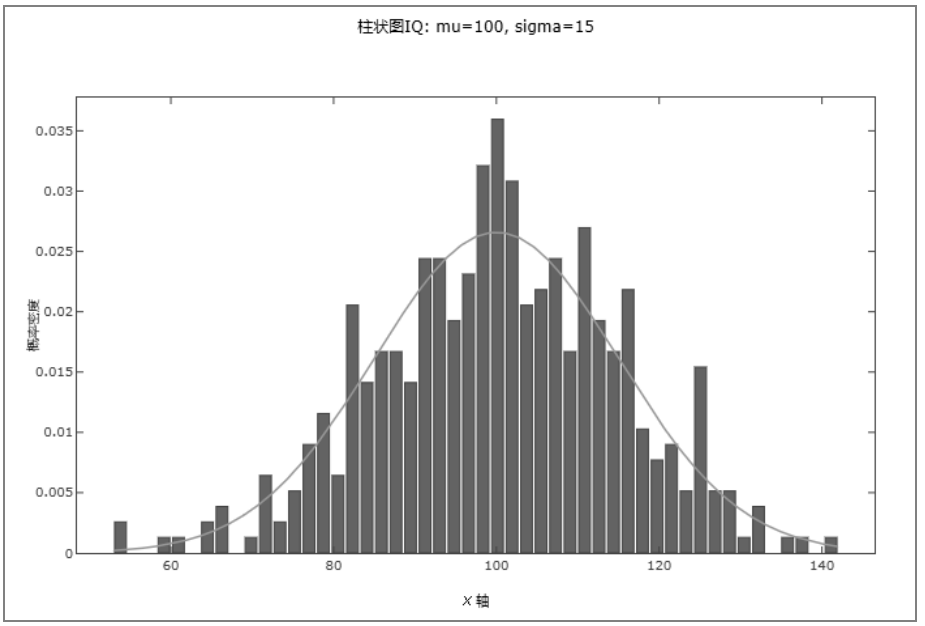

图 6-10　正态分布曲线绘图

6.9　散点图

1. 散点图一

散点图是指在回归分析中，数据点在直角坐标系平面上的分布图。散点图表示因变量随自变量变化的大致趋势，据此可以选择合适的函数对数据点进行拟合。

用两组数据构成多个坐标点，考察坐标点的分布，用来判断两个变量之间是否存在某种关联或用来总结坐标点的分布模式。散点图将序列显示为一组点，值由点在图表中的位置表示，类别由图表中的不同标记表示。散点图通常用于跨类别的聚合数据。

本案例见文件 scatter_1.py，代码如下。

```
import numpy as np
#使用 import 导入模块 NumPy
import matplotlib.pyplot as plt
#使用 import 导入模块 matplotlib.pyplot
import plotly as py
```

```
    # 导入 Plotly 库并命名为 py
    # -------------pre def
    pympl = py.offline.plot_mpl
    # 配置中文显示
    plt.rcParams['font.family'] = ['SimHei']  # 用来正常显示中文标签
    plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
    fig, ax = plt.subplots()
    #生成测试数据
    x = np.arange(1,30)
    y =np.sin(x)
    ax1 = fig.add_subplot(111)
    #设置标题
    ax1.set_title('散点图')
    #设置 X 轴标签
    plt.xlabel('X')
    #设置 Y 轴标签
    plt.ylabel('Y')
    #画散点图
    lValue = x
    ax1.scatter(x,y,c='r',s= 100,linewidths=lValue,marker='o')
    #设置图标
    plt.legend('x1')
    plot_url = pympl(fig,filename=r'tmp/scatter_1.html', show_link=False,
resize=True)
    #离线画图
```

scatter 函数的用法是 scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, *, data=None, **kwargs)

scatter 函数中比较常用的参数解释如下。

- x,y：形如 shape(n,)的数组，可选值。
- s：点的大小（也就是面积），默认值为 20。
- c：点的颜色或颜色序列，默认为蓝色，其他如 c = 'r' (red)、c = 'g' (green)、c = 'k' (black)、c = 'y'(yellow)。
- marker：形状，可选值，默认是圆。

绘图结果如图 6-11 所示。

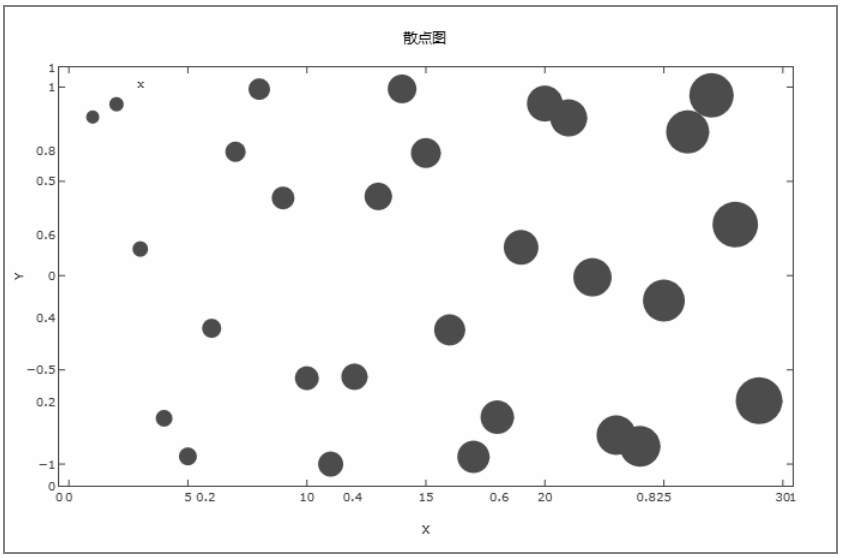

图 6-11　散点图一绘图结果

2．散点图二

本案例见文件 scatter_2.py，代码如下。

```
from numpy.random import rand
#使用 import 导入模块 NumPy
import matplotlib.pyplot as plt
#使用 import 导入模块 matplotlib.pyplot
import plotly as py
# 导入 Plotly 库并命名为 py
# -------------pre def
pympl = py.offline.plot_mpl
# 配置中文显示
plt.rcParams['font.family'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

fig, ax = plt.subplots()
for color in ['red', 'green', 'blue']:
    n = 500
    x, y = rand(2, n)#生成 2*n 的矩阵
    ax.scatter(x, y, c=color, label=color,
```

```
                alpha=0.3, edgecolors='none')
    #设置每个点的颜色，x、y 分别代表横、纵坐标，通过横、纵坐标确定点的位置；c 表示点的颜色；s 表示点的大小；alpha 表示点的透明度，1 是不透明，0 是透明
    ax.legend()
    ax.grid(True)
    #设置网格
    plot_url = pympl(fig,filename=r'tmp/scatter_2.html', show_link=False, resize=True)
    #离线画图
```

绘图结果如图 6-12 所示。

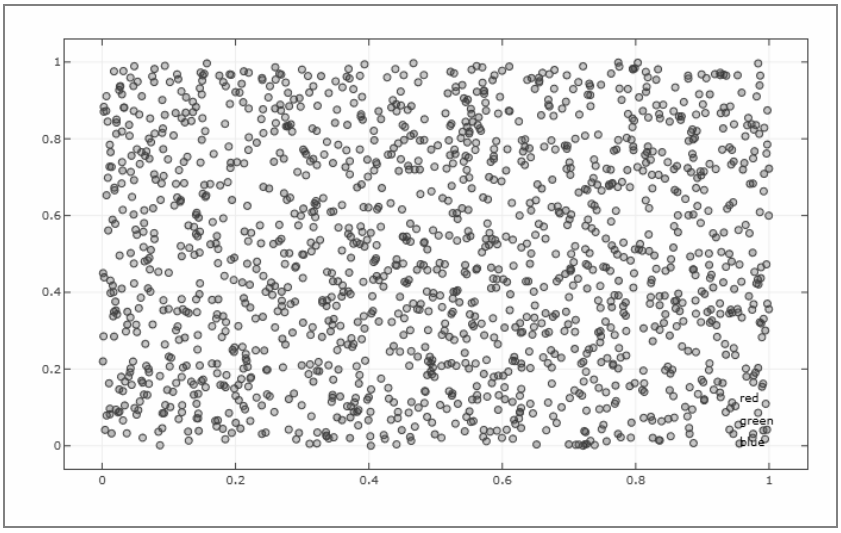

图 6-12　散点图二绘图结果

6.10　柱状图

1. 柱状图一

柱状图是一种以长方形的长度为变量的表达图形的统计报告图，由一系列高度不等的纵向条纹表示数据分布的情况，用来比较两个或以上的价值（不同时间或不同条件）。如果只有一个变量，则通常用于较少的数据集分析。柱状图也可横向排列，或者用多维方式表达。

本案例见文件 bar_1.py，代码如下。

```
    import numpy as np
    #使用 import 导入模块 NumPy
    import matplotlib.pyplot as plt
    #使用 import 导入模块 matplotlib.pyplot
    import plotly as py
     # 导入 Plotly 库并命名为 py
    # -------------pre def
    pympl = py.offline.plot_mpl
    # 配置中文显示
    plt.rcParams['font.family'] = ['SimHei']  # 用来正常显示中文标签
    plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号


    plt.figure(figsize=(7,5))


    X = np.arange(1,6)
    #X 是柱的个数
    # numpy.random.uniform(low=0.0, high=1.0, size=None), normal
    #uniform 是均匀分布的随机数，normal 是正态分布的随机数，0.5～1 均匀分布的数一共有
n 个
    Y1 = np.random.uniform(0.5,1.0,5)
    Y2 = np.random.uniform(0.5,1.0,5)
    plt.bar(X,Y1,width = 0.35,facecolor = 'lightskyblue',edgecolor =
'white')
    #width 为柱的宽度
    plt.bar(X+0.35,Y2,width = 0.35,facecolor = 'yellowgreen',edgecolor =
'white')
    #水平柱状图 plt.barh，属性中宽度 width 变成了高度 height
    #facecolor 是柱状图里填充的颜色
    #edgecolor 是边框的颜色
    fig = plt.gcf()
    plot_url = pympl(fig,filename=r'tmp/bar_1.html',
show_link=False,resize=True)
    #离线画图
```

bar 函数的使用方法是：bar（left, height, width, bottom=None, **kwargs）。

事实上，left、height、width、bottom 这四个参数确定了柱体的位置和大小。默认情况下，left 为柱体的居中位置（可以通过 align 参数改变 left 值的含义），即：

- （left - width / 2, bottom）为左下角位置；
- （left + width / 2, bottom + height）为右上角位置。

绘图结果如图 6-13 所示。

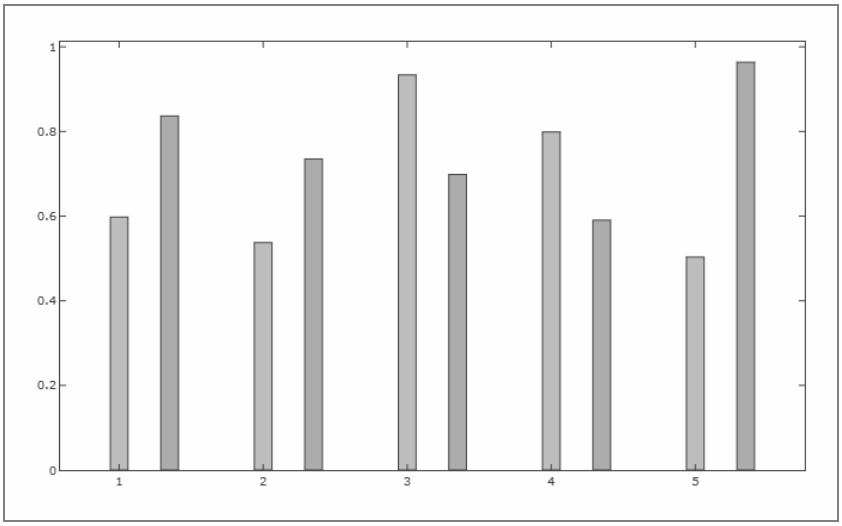

图 6-13　柱状图一绘图结果

2．柱状图二

本案例见文件 bar_2.py，代码如下。

```
import numpy as np
#使用 import 导入模块 NumPy
import matplotlib.pyplot as plt
#使用 import 导入模块 matplotlib.pyplot
import plotly as py
# 导入 Plotly 库并命名为 py
# -------------pre def
pympl = py.offline.plot_mpl
# 配置中文显示
plt.rcParams['font.family'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
```

```
    np.random.seed(19680801)
    #生成随机种子
    n_bins = 10
    x = np.random.randn(1000, 3)
    #生成 1000*3 的矩阵
    fig, axes = plt.subplots(nrows=2, ncols=2)
    ax0, ax1, ax2, ax3 = axes.flatten()

    colors = ['red', 'tan', 'lime']
    #设置颜色
    ax0.hist(x, n_bins, normed=1, histtype='bar', color=colors,
label=colors)
    #设置柱状图的信息
    ax0.legend(prop={'size': 10})
    ax0.set_title('图例 bar')
    #添加标题
    ax1.hist(x, n_bins, normed=1, histtype='bar', stacked=True)
    #设置柱状图的信息

    ax1.set_title('堆积 bar')
    #添加标题
    ax2.hist(x, n_bins, normed=1, histtype='bar', stacked=True, fill=False)
    #设置柱状图信息
    ax2.set_title('堆积 bar（非填充）')
    #添加标题
    x_multi = [np.random.randn(n) for n in [10000, 5000, 2000]]
    ax3.hist(x_multi, n_bins, histtype='bar')
    #设置柱状图信息
    ax3.set_title('不同样本容量')
    #添加标题
    fig.tight_layout()
    #使图形紧凑
    fig1= plt.gcf()
    plot_url = pympl(fig1,filename=r'tmp/bar_2.html',
show_link=False,resize=True)
    #离线画图
```

绘图结果如图 6-14 所示。

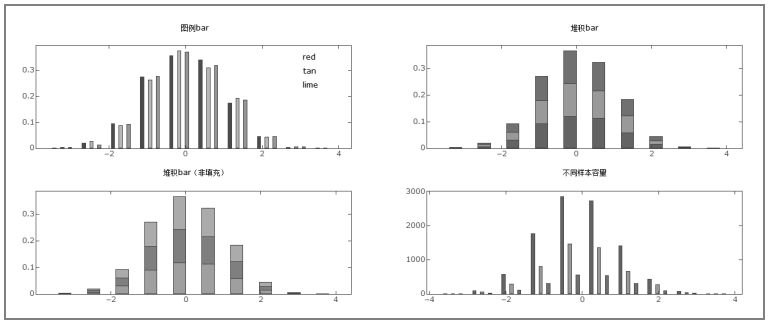

图 6-14　柱状图二绘图结果

6.11　更多扩展

读者可以打开网址 https://plot.ly/matplotlib/，获取 Matplotlib 与 Plotly 结合的所有官方案例（如图 6-15 所示）。与标准的 Plotly 实例相比，内容少了一些，但是胜在可以和 Matplotlib 无缝结合，而且绝大部分绘图类型都能找到，特别适合对 Matplotlib 熟悉的读者。

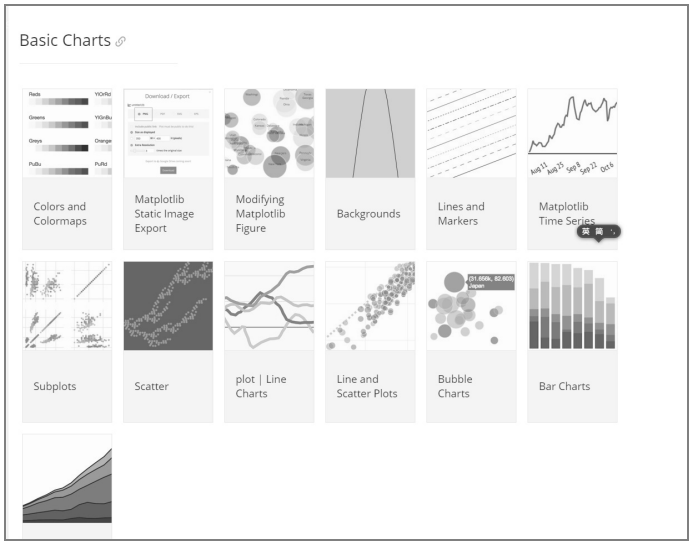

图 6-15　官方案例

第 7 章

Plotly 与网页开发

7.1 Plotly在Django中的应用

Django 是用 Python 语言开发的一个免费开源的 Web 框架，用于快速搭建高性能、优雅的网站，通过 Django 可以建立一个高性能的 Web 应用而只花费最少的时间和精力。

Django 中提供开发网站经常用到的模块，常见的代码都为用户写好了，通过减少编写重复的代码，用户能够专注于 Web 应用上关键性的东西。为了达到这个目标，Django 提供了通用 Web 开发模式的高度抽象和频繁进行编程作业的快速解决方法，并为“如何解决问题”提供了清晰明确的约定。Django 的理念是 DRY（Don't Repeat Yourself），鼓励用户快速开发！

对于本章内容，首先假设读者已经对 Django 和 Plotly 有了一些基本的了解，本章内容的重点是把 Plotly 嵌入到 Django 中，并不会过多地介绍 Django 与 Plotly 的具体细节内容。

7.1.1 安装环境搭建

Django 的安装配置非常简单，如果本地计算机已经安装了 Django，则不再进行

安装；如果没有安装，则可以通过下面的命令安装最新版本的 Django，只需要简单的一行代码：

```
pip install django
```

7.1.2　安装环境测试

安装好 Django 安装包以后，需要测试安装环境是否搭建成功。首先，可以通过下面的命令创建一个新的 Django 项目，此处命名为 ploty_and_django，代码如下。

```
django-admin startproject plotly_and_django
```

运行完上面的命令后，会得到下面的层级结构。

```
plotly_and_django/
    manage.py
    plotly_and_django/
        __init__.py
        settings.py
         urls.py
         wsgi.py
```

这些文件的具体含义如下。

（1）manage.py：这是项目交互的命令行实用程序，是 django-admin.py 工具周围的薄型包装，不需要编辑此文件。

（2）plotly_and_django/：用户的项目文件夹，包含以下文件。

- __init__.py：一个空的文件，用来告诉 Python 把这个文件当作 Python 的包对待。
- settings.py：用于调整项目的设置和参数。
- urls.py：用来储存项目中的 URL，Django 会通过正则表达式来匹配 URL。
- wsgi.py：里面包含了通过 WSGI 方式运行 Django 项目的一系列参数。

接着，在本地计算机上通过下面的命令运行测试服务器：

```
python manage.py runserver
```

注意

确保已经进入项目的文件夹，如果没有运行这条命令，则会直接运行后面的

命令：

```
cd plotly_and_django
```

这时会看到类似下面的输出结果：

```
Performing system checks...

System check identified no issues (0 silenced).
August 13, 2017 - 16:41:39
Django version 1.11, using settings 'plotly_and_django.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

打开浏览器，在地址栏中输入 http://127.0.0.1:8000/，如果一切正常，就会看到 Django 欢迎页面，如图 7-1 所示。

Welcome to Django
127.0.0.1:8000
Search
It worked!
Congratulations on your first Django-powered page.
Next, start your first app by running python manage.py startapp [app_label].
You're seeing this message because you have DEBUG = True in your Django settings file and you haven't configured any URLs. Get to work!

图 7-1　Django 欢迎页面

于是，我们成功配置了 Django 的安装环境。

注意

本案例不需要和数据库连接，如果今后的项目需要和数据库连接，请确保每次修改数据表格定义时使用下面的命令：

```
python manage.py migrate
```

执行上面的命令后，就会看到类似下面的输出：

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
```

```
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying sessions.0001_initial... OK
```

7.1.3　入门案例一

下面把 Django 和 Plotly 进行简单的结合，具体的原理是在前端访问服务器时，Django 会在服务器端生成包含 Plotly 绘图结果的 HTML 文件，然后返回给浏览器，从而实现数据的可视化，具体步骤如下。

（1）新建一个应用

通过下面的命令在项目中新建一个应用，命名为 visual_plotly。

```
python manage.py startapp visual_plotly
```

这时项目文件夹中会多出一个名称为 visual_plotly 的文件夹。一个 Django 项目可以有多个应用，每个应用都有一个这样的文件夹与之对应。visual_plotly 文件夹中会自动生成下面的文件目录结构：

```
visual_plotly/
  __init__.py
  migrations/
    __init__.py
  admin.py
  models.py
  tests.py
  views.py
```

对上面的目录结构解释如下。

- migrations：这个文件主要包含该应用和数据库相关的所有变化。Django 记录这些变化，就能保证数据库与 Django 同步更改。
- admin.py：在该文件中可以决定应用是否在 Django 的 Admin 页面注册，Django 的 Admin 页面是 Django 自带的一个管理平台，功能比较强大，也很好用，这里不做过多说明。
- models.py：用于定义应用中用到的模型，模型实际上就是一些数据表格，这些表格能和数据库的表格一一对应。Django 通过这种方式提供了统一的接口，这样即使更换了数据库，比如从 MySQL 到 Oracle，项目中的代码都可以不用该数据表格，只要模型没有改变。
- tests.py：可以添加一些测试代码用来测试应用。
- views.py：存放处理 HTTP 请求的内容，本案例将会在这里添加 Plotly 代码，实现返回 HTML 页面。

（2）创建 templates 文件夹

在 plot_and_djano 文件夹下面创建 templates 文件夹，然后打开 plot_and_django/settings.py 文件，修改里面关于 TEMPLATES 的部分，这里只需要修改关于“DIRS”的那一项。为了便于比较，整段代码如下。

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates'),],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

在 templates 文件夹中新建 index.html 文件，并输入下面的代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Plotly 嵌入 Django 案例</title>
</head>
<body>

{% if graph %}
<div style="width:600;height:500">
{{ graph|safe }}
</div>
{% endif %}

</body>
</html>
```

（3）修改 urls.py 文件

对 plotly_and_django/urls.py 文件做如下修改：

```
from django.conf.urls import url, include

urlpatterns = [
    url(r'', include('visual_plotly.urls')),
]
```

这里把首页指向了新建的 visual_plotly 应用，其实完全可以对某个特殊的地址进行绑定，在这里不做过多介绍。

（4）新建 urls.py 文件

在 visual_plotly 中新建 urls.py 文件，并添加下面的代码：

```
from .views import plotly_view
from django.conf.urls import url


urlpatterns = [
    url(r'^$', plotly_view),
]
```

对于分配到的请求，调用 views 文件里面的 plotly_view 函数。

（5）修改 views.py 文件

修改 visual_plotly/views.py 文件，代码如下。

```
    from django.shortcuts import render

    import plotly as py
    import plotly.graph_objs as go
    pyplt = py.offline.plot

    def plotly_view(request):

        trace0 = go.Bar(
            x=['Product A', 'Product B', 'Product C'],
            y=[20, 14, 23],
            text=['27% market share', '24% market share', '19% market share'],
            marker=dict(
                color='rgb(158,202,225)',
                line=dict(
                    color='rgb(8,48,107)',
                    width=1.5,
                )
            ),
            opacity=0.6
        )

        data = [trace0]
        layout = go.Layout(
            title='January 2013 Sales Report',
        )

        fig = go.Figure(data=data, layout=layout)
        div = pyplt(fig, output_type='div', auto_open=False,
show_link=False)
        context = {}
        context['graph'] = div
        return render(request, 'index.html', context=context)
```

在这个函数中可以嵌套不同的 Plotly 绘图代码，无论数据是从数据库动态调取的，还是结合其他复杂逻辑的，比如分情况生成不同的图，这种框架都能将 Plotly 绘图嵌套在网页中。

（6）启动本地测试服务器

在命令行窗口输入如下代码：

```
python manage.py runserver
```

打开浏览器，在地址栏中输入网址 http://127.0.0.1:8000/，就会看到通过 Plotly 生成的网页，如图 7-2 所示。

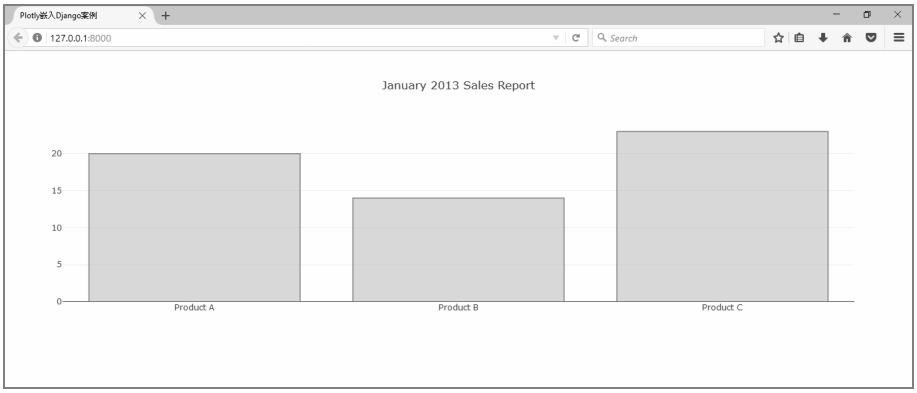

图 7-2　通过 Plotly 生成的网页

由此，我们实现了把 Plotly 嵌入到 Django 网页开发的第一个应用。

7.1.4　入门案例二

前面我们完成了简单的入门案例，注意这个案例有一个缺点，那就是传入的 div 参数有点大，约有 2MB。如果一个网页中只有一两个这种绘图，用这个案例的方法是可以接受的，但是如果有多个绘图，用这个案例的方法就不太现实了，因为会导致页面数据量太大而给服务器造成很大的压力，下面来看一下出现这个问题的原因。

首先，要分析一下传递的 div 参数是什么。打开 test_memory1.py 文件并运行，代码如下。

```
import plotly as py
import plotly.graph_objs as go

pyplt = py.offline.plot

# test memory1

trace0 = go.Bar(
    x=['Product A', 'Product B', 'Product C'],
    y=[20, 14, 23],
    text=['27% market share', '24% market share', '19% market share'],
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data = [trace0]
layout = go.Layout(
    title='January 2013 Sales Report',
)

fig = go.Figure(data=data, layout=layout)
div = pyplt(fig, output_type='div', auto_open=False, show_link=False)

if __name__ == '__main__':
    import sys
    print('参数 div 占用内存大小为 %d bytes'%sys.getsizeof(div))
    with open('div1.txt', 'w') as file:
        file.write(div)
```

输出结果如下：

```
参数 div 占用内存大小为 2341594 bytes
```

这个 div 参数非常大，约有 2MB，它本质上是 plotly.js 与调用 plotly.js 绘图代码的结合体，打开 div1.txt 就可以看到这个文件前面的内容是 plotly.js 文件，后面的几行内容才是绘图的代码。我们实际需要的是后面的代码，对于前面的 plotly.js，只需要在 HTML 文件中引用一次，没必要每次绘图都重新引用。

因此，需要剥离 plotly.js，方法很简单，只需要在原来的基础上加上参数 include_plotlyjs=False 就行了，代码如下（见文件 test_memory2.py）。

```
    import plotly as py
    import plotly.graph_objs as go

    pyplt = py.offline.plot

    # test memory1

    trace0 = go.Bar(
        x=['Product A', 'Product B', 'Product C'],
        y=[20, 14, 23],
        text=['27% market share', '24% market share', '19% market share'],
        marker=dict(
            color='rgb(158,202,225)',
            line=dict(
                color='rgb(8,48,107)',
                width=1.5,
            )
        ),
        opacity=0.6
    )

    data = [trace0]
    layout = go.Layout(
        title='January 2013 Sales Report',
    )

    fig = go.Figure(data=data, layout=layout)
    div = pyplt(fig, output_type='div', auto_open=False, show_link=False,
include_plotlyjs=False)
```

```
if __name__ == '__main__':
    import sys
    print('参数 div 占用内存大小为 %d bytes'%sys.getsizeof(div))
    with open('div2.txt', 'w') as file:
        file.write(div)
```

运行结果如下：

```
参数 div 占用内存大小为 695 bytes
```

打开 div2.txt 文件，会看到如下内容：

```
<div id="eae67571-a5ea-4204-a119-942c560d0f2e" style="height: 100%;
width: 100%;" class="plotly-graph-div"></div><script
type="text/javascript">window.PLOTLYENV=window.PLOTLYENV ||
{};window.PLOTLYENV.BASE_URL="https://plot.ly";Plotly.newPlot("eae67571-
a5ea-4204-a119-942c560d0f2e", [{"y": [20, 14, 23], "opacity": 0.6, "text":
["27% market share", "24% market share", "19% market share"], "type": "bar",
"x": ["Product A", "Product B", "Product C"], "marker": {"line": {"width":
1.5, "color": "rgb(8,48,107)"}, "color": "rgb(158,202,225)"}}], {"title":
"January 2013 Sales Report"}, {"linkText": "Export to plot.ly", "showLink":
false})</script>
```

这才是 Plotly 真正的代码，实际上我们只需要在 HTML 文件中引用这段代码就行了，plotly.js 可以通过其他方式导入。

下面讲解如何将剥离 plotly.js 后的 div 嵌入到 Django 中。我们这次的案例建立在“入门案例一”的基础上，只需要进行一些改动，步骤如下。

（1）复制 plotly.min.js 文件

在 plotly_and_django 目录下新建文件夹 static，复制 plotly.min.js 文件（笔者的 Python 环境目录为：d:\Anaconda3\Lib\site-packages\plotly\package_data\plotly.min.js）到 static 文件夹，目录结构如下：

```
D:\zw_own\MyPlotlyBook\plotly_and_django>tree /F
D:.
|  db.sqlite3
|  manage.py
|
```

```
├──plotly_and_django
│   │  settings.py
│   │  urls.py
│   │  wsgi.py
│   │  __init__.py
│   │
├──static
│      plotly.min.js
│
├──templates
│      index.html
│
└──visual_plotly
    │  admin.py
    │  apps.py
    │  models.py
    │  tests.py
    │  urls.py
    │  views.py
    │  __init__.py
    │
    ├──migrations
```

（2）修改 setting.py 文件

修改 plotly_and_django/settings.py 文件，添加 static 文件路径，方便 Django 查找 plotly.min.js，代码如下。

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.11/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS =(
    os.path.join(BASE_DIR,'static'),
)
```

（3）修改 urls.py 文件

修改 visual_plotly\urls.py 文件，添加如下信息（意思是新建一个页面如 http://127.0.0.1:8000/demo2，指向 plotly_view2），代码如下。

```
    from .views import plotly_view,plotly_view2
    from django.conf.urls import url


    urlpatterns = [
        url(r'^$', plotly_view),
        url(r'^demo2', plotly_view2),
    ]
```

（4）修改 views.py 文件

修改 visual_plotly/views.py 文件，添加如下函数：

```
    def plotly_view2(request):
        trace0 = go.Bar(
            x=['Product A', 'Product B', 'Product C'],
            y=[20, 14, 23],
            text=['27% market share', '24% market share', '19% market share'],
            marker=dict(
                color='rgb(158,202,225)',
                line=dict(
                    color='rgb(8,48,107)',
                    width=1.5,
                )
            ),
            opacity=0.6
        )

        data = [trace0]
        layout = go.Layout(
            title='January 2013 Sales Report——No plotly.js',
        )

        fig = go.Figure(data=data, layout=layout)
        div = pyplt(fig, output_type='div', auto_open=False, show_link=False,
include_plotlyjs=False)
        context = {}
        context['graph2'] = div
        return render(request, 'index.html', context=context)
```

（5）修改 index.html 文件

修改 templates/index.html 文件，代码如下。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Plotly 嵌入 Django 案例</title>
</head>
<body>

<!-- load plotly.js -->
<script src="static/plotly.min.js"></script>

{% if graph %}
<div style="width:600;height:500">
{{ graph|safe }}
</div>
{% endif %}


{% if graph2 %}
<div>
{{ graph2|safe }}
</div>
{% endif %}

</body>
</html>
```

与之前的 index.html 文件相比，添加了如下代码：

```
<!-- load plotly.js -->
<script src="static/plotly.min.js"></script>

{% if graph2 %}
<div>
{{ graph2|safe }}
```

```
    </div>
    {% endif %}
```

（6）运行测试

在命令行窗口输入如下代码：

```
python manage.py runserver
```

打开浏览器，在地址栏中输入 http://127.0.0.1:8000/demo2，将会看到通过 Plotly 生成的新页面（如果输入 http://127.0.0.1:8000，打开的页面就会和原来的页面一样），如图 7-3 所示。

图 7-3　通过 Plotly 生成的新页面

至此，我们成功地把脱离 plotly.js 后的 div 嵌入到 Django 中，这种嵌入方式所嵌入的 div 内容比较小且逻辑清晰，比较贴近实际，因此后续讲解的案例都会以这种方式为主。

7.1.5　更多案例扩展

上面的案例虽然看起来简单，却解决了 Plotly 与 Django 交互时最大的难题，成功地把 Plotly 嵌入到 Django 中。这里建立了 Plotly 嵌入到 Django 中的简单框架，从 Plotly 的角度来说，如果想要把更多的 Plotly 绘图嵌入到 Django 中，则需要修改 Plotly 对应的代码；从 Django 的角度来说，如果想要在网页中呈现更多的内容，则需要调整网页的结构，这就需要对 Django 对应的代码进行调整。本节内容重点讨论前者，下一节会进一步讨论后者。

从前面的内容可以看出，Plotly 部分的代码主要放在 visual_plotly/views.py 文件中，代码如下。

```
    from django.shortcuts import render

    import plotly as py
    import plotly.graph_objs as go
    pyplt = py.offline.plot

    def plotly_view(request):

        trace0 = go.Bar(
            x=['Product A', 'Product B', 'Product C'],
            y=[20, 14, 23],
            text=['27% market share', '24% market share', '19% market share'],
            marker=dict(
                color='rgb(158,202,225)',
                line=dict(
                    color='rgb(8,48,107)',
                    width=1.5,
                )
            ),
            opacity=0.6
        )

        data = [trace0]
        layout = go.Layout(
            title='January 2013 Sales Report',
        )

        fig = go.Figure(data=data, layout=layout)
        div = pyplt(fig, output_type='div', auto_open=False, show_link=
False,include_plotlyjs=False)
        context = {}
        context['graph'] = div
        return render(request, 'index.html', context=context)
```

上面的代码我们非常熟悉，基本上与本书之前讲解的 Plotly 绘图部分的内容相同。在这些代码中，最关键的部分是下面几行代码：

```
    fig = go.Figure(data=data, layout=layout)
    div = pyplt(fig, output_type='div', auto_open=False,
```

```
show_link=False,include_plotlyjs=False)
    context = {}
    context['graph'] = div
    return render(request, 'index.html', context=context)
```

这几行代码实现的功能是把 Plotly 的离线绘图结果存储到一个 div 参数（pyplt 返回的结果）里面，这个变量的内容是调用 plotly.js 进行绘图，我们把它放在 context 字典里，传递给 Django 的 index.html 模板，最后在 index.html 模板中显示这个绘图结果。

与之前 Plotly 的绘图函数不同的是，我们这里把结果以.js 的形式存储到一个变量 div 里面（output_type='div'的作用），并设置默认不在网页中打开（auto_open=False 的作用），同时输出结果中不包含 plotly.js（include_plotlyjs=False 的作用）。

从上面的代码中可以看出，我们完全可以修改上面的绘图代码，换成本书前面章节所讲过的绘图代码和自己喜欢的 Plotly 绘图代码，从而把更多的 Plotly 嵌入到 Django 中。这些事情对读者来说并不是太难，读者可以自行完成。

7.1.6 应用案例一

接下来要做的事情就是在网页上呈现更多的信息，最简单的应用案例是在 index.html 模板中添加一些信息，组成一个简单的网页。本案例用一个简单的网页实现对房价的绘图分析，这个案例相对于前面的案例主要修改的是 index.html 文件，其代码（见文件 plotly_django_app1/templates/index.html）如下。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Plotly 嵌入 Django 案例</title>
    <style type="text/css">
    div.content {
        padding: 1em 5em;
    }
    div.graph {
        margin: 10px auto;
```

```
            width: 70%;
        }

        </style>
    </head>
    <body>

    <!-- load plotly.js -->
    <script src="static/plotly.min.js"></script>

    <div class='content'>
        <h2>用 Python 爬虫抓取最新的房价数据</h2>
        <div>
            <p>
             部分城市的房价迅速上涨，多地出台楼市限购政策。未来房价走势如何？由于影响因素的多样性，目前还没有发现任何一种研究分析能够确切地预测，但我们可以自己动手从网络上抓取房价的最新数据，密切关注最新的房价信息，结合实际情况做出自己的判断。
            </p>
            <p>
                下面数据化分析以链家网为例，利用 Python 爬虫抓取三种户型的平均房价（千元/平方米）:
            </p>
        </div>

        {% if graph %}
        <div class='graph'>
        {{ graph|safe }}
        </div>
        {% endif %}

        <div>
            <p>从图上可以看出，C 类户型的平均房价最高，最受欢迎，其次是 A 类户型，最后是 B 类户型</p>
        </div>
    </div>


    </body>
    </html>
```

运行 python manage.py runserver，在浏览器的地址栏中输入 http://127.0.0.1:8000，结果如图 7-4 所示。

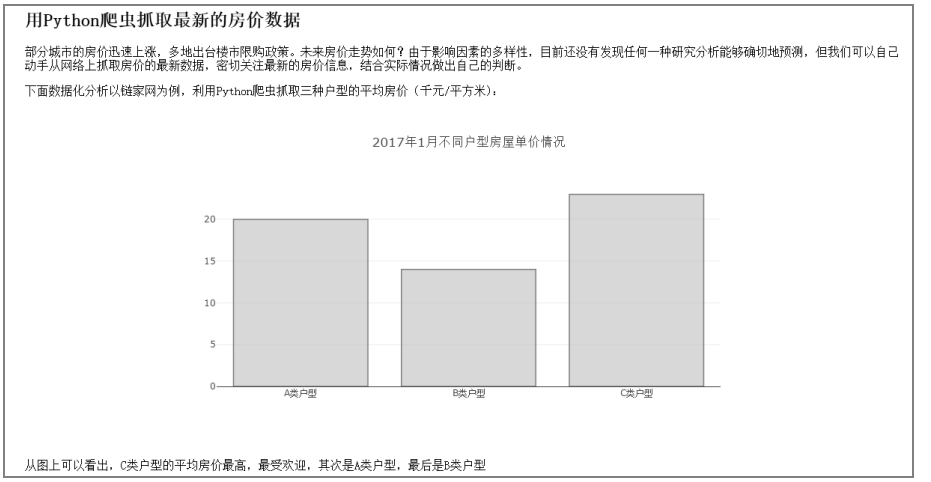

图 7-4　应用案例一的绘图结果

这只是一个简单的应用案例，这个案例虽然简单，但是对于复杂的案例，这种方法同样适用。接下来为读者介绍一个相对综合的案例。

7.1.7　应用案例二

在实际生产环境中，我们所涉及的模板 templates/index.html 与静态文件 static/plotly.min.js 是很复杂的，并且也不止一个，但是 Django 调用模板的原理是一样的，不会因为其复杂而改变调用它们的方法，因此在真实的生产环境中，Plotly 嵌入到 Django 中的方法与之前的方法相比没有什么不同。

由于真实的 Plotly 商业案例特别复杂、难以理解，并且涉及商业机密，因此不适合作为新手的入门案例，本书选择了相对简单的应用案例。幸运的是，笔者在网上找到了一些适合 Plotly 实战应用的 HTML 模板，这些模板完全可以满足我们实际的需求，因此拿来做案例再好不过。Web 开发并不是本书的重点，本书重点内容是 Plotly 在 Web 开发中的应用，因此本节只介绍 Plotly 嵌入 Web 的相关内容。

本案例见文件 plotly_django_app2，主要目录结构如下（删掉部分是不重要的文

件结构)：

```
F:\Project\plotly_django_app2>tree /F
F:.
|  db.sqlite3
|  manage.py
├──plotly_django
|   |  settings.py
|   |  urls.py
|   |  wsgi.py
|   |  __init__.py
├──plotly_django_app2
|   |  admin.py
|   |  apps.py
|   |  chart_plot.py
|   |  models.py
|   |  tests.py
|   |  views.py
|   |  __init__.py
|   |
|   ├──dat
|   |      day01.csv
|   |
|   ├──migrations
|   |   |  __init__.py
├──static
|   |  plotly.min.js
|   |
|   ├──bootstrap
|
├──templates
|      bookings.html
|      calendar.html
|      charts.html
|      documentation.html
|      file_managers.html
|      finances.html
|      form_demo.html
|      form_elements.html
```

```
    |      form_validator.html
    |      gallery.html
    |      index.html
    |      login.html
    |      pages.html
    |      product_edit.html
    |      tables.html
    |      ui.html
```

本案例的文档结构看上去复杂，其实在实际调用中与之前的案例方法是一样的，与之前的案例内容相比，无非是 urls.py、view.py 内容多了一些，并且 template 模板的 HTML 代码多了一些而已，下面重点介绍其中的变化。

（1）在 plotly_django/urls.py 文件中，将每个 URL 作为一个请求传递给 views.py，代码如下。

```
from django.conf.urls import url
from django.contrib import admin
from plotly_django_app2 import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^charts', views.charts),
    url(r'^index', views.index),
    url(r'^bookings', views.bookings),
    url(r'^calendar', views.calendar),
    url(r'^documentation', views.documentation),
    url(r'^file_managers', views.file_managers),
    url(r'^finances', views.finances),
    url(r'^form_demo', views.form_demo),
    url(r'^form_elements', views.form_elements),
    url(r'^form_validator', views.form_validator),
    url(r'^gallery', views.gallery),
    url(r'^login', views.login),
    url(r'^pages', views.pages),
    url(r'^product_edit', views.product_edit),
    url(r'^tables', views.tables),
    url(r'^ui', views.ui),
    url(r'', views.start), # 除上面的规则外，其他规则都会指向 views.start
]
```

其中，需要注意的是最后一个 URL：

```
url(r'', views.start), # 除上面的规则外，其他规则都会指向 views.start
```

这个 URL 表示除上面的规则外，其他规则都会指向 views.start。views.start 是一个页面跳转的代码，不是直接返回某个 HTML 模板，这一点会在随后的 views.py 文件中说明。

（2）在 plotly_django_app2/view.py 文件中，针对每个请求，返回相应的 HTML 模板或跳转到指定的 URL，代码如下。

```
import os
from django.shortcuts import redirect
from django.shortcuts import render

CURRENT_PATH = os.path.dirname(os.path.abspath(__file__))
# Create your views here.
from plotly_django_app2.chart_plot import ChartPlot
chart = ChartPlot()
context = {}

def index(request):
    context['graph_range_slide'] = chart.range_slide()
    return render(request, 'index.html', context=context)

def start(request):
    return redirect(r'/index.html')

def charts(request):
    context['graph_basic_plot']= chart.basic_plot()
    context['graph_line_full'] = chart.line_full()
    context['graph_bar_chart'] = chart.bar_chart()
    context['graph_bonus_chart'] = chart.bonus_chart()
    context['graph_pie_chart'] = chart.pie_chart()
    context['graph_stacked_bar'] = chart.stacked_bar()
    context['graph_horizontal_chart'] = chart.horizontal_chart()
    return render(request, 'charts.html', context=context)


def finances(request):
```

```
    return render(request, 'finances.html')


def ui(request):
    return render(request, 'ui.html')


def tables(request):
    return render(request, 'tables.html')


def form_demo(request):
    return render(request, 'form_demo.html')


def form_elements(request):
    return render(request, 'form_elements.html')


def bookings(request):
    return render(request, 'bookings.html')


def calendar(request):
    return render(request, 'calendar.html')


def documentation(request):
    return render(request, 'documentation.html')


def file_managers(request):
    return render(request, 'file_managers.html')


def form_elements(request):
    return render(request, 'form_elements.html')
```

```
def form_validator(request):
    return render(request, 'form_validator.html')


def gallery(request):
    return render(request, 'gallery.html')


def login(request):
    return render(request, 'login.html')

def pages(request):
    return render(request, 'pages.html')

def product_edit(request):
    return render(request, 'product_edit.html')
```

这里需要注意以下 3 点。

① ChartPlot 是 plotly_django_app2/chart_plot.py 文件中的类，类中的每个函数都是 Plotly 的一个绘图函数，其返回值是一个去掉 plotly.js 的 div 参数。

② start 函数返回的 redirect 是一个页面跳转的函数，表示如果触发该请求，则把它跳转到 index.html 页面。

```
def start(request):
    return redirect(r'/index.html')
```

③ 我们只对 charts.html 与 index.html 这两个模板文件的绘图代码进行修改，稍后可以检测这两个图修改后的效果。

（3）按照之前的方式修改 templates/index.html 与 templates/charts.html 文件，传递 div 参数。以 index.html 为例，主要的工作是在原来的模板上添加如下代码。

```
    <!-- load plotly.js -->
<script src="static/plotly.min.js"></script>

       {% if graph_range_slide %}
          <div>
             {{ graph_range_slide|safe }}
          </div>
```

```
        {% endif %}
```

这种方法和之前学习的内容没有什么区别。

运行下面的代码，8001 表示端口号为 8001（默认为 8000，注意与之前的端口号不同）。

```
python manage.py runserver 8001
```

打开网页 http://127.0.0.1:8001/index.html，会看到如图 7-5 所示的页面。

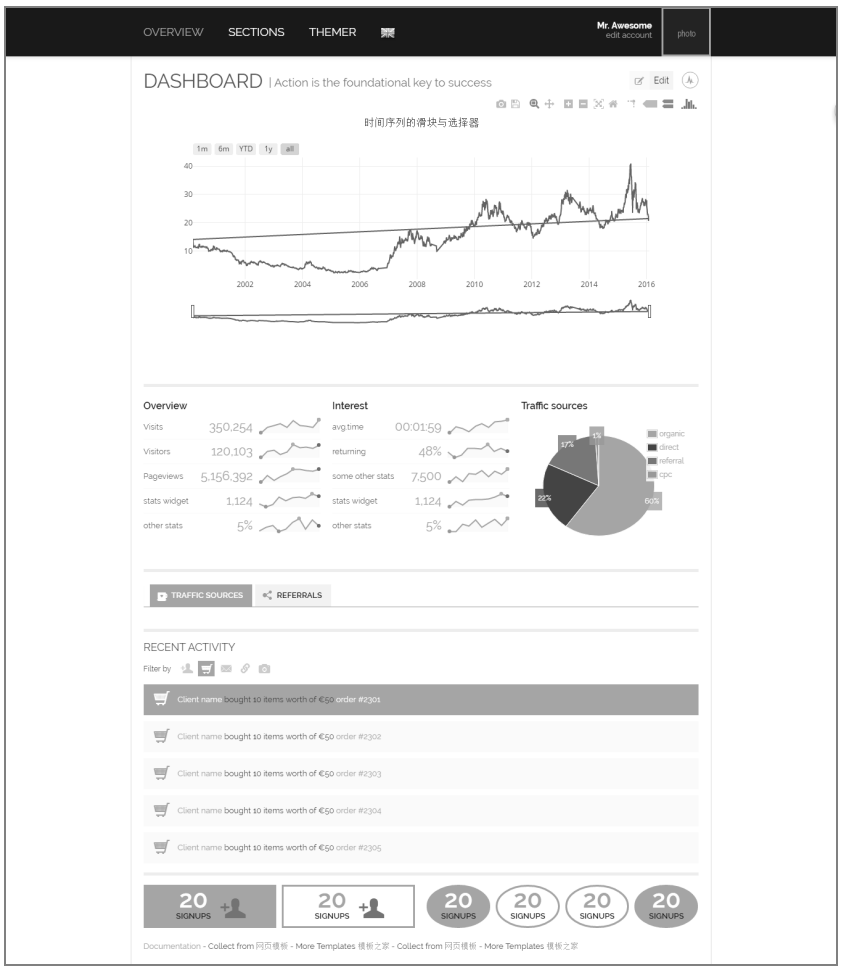

图 7-5　index.html 页面

图 7-5 中第一个时间序列的图就是 Plotly 的绘图效果。

打开网页 http://localhost:8001/charts.html，运行效果如图 7-6 和图 7-7 所示。

图 7-6　charts.html 页面 1

图 7-7　charts.html 页面 2

由于其他页面的效果与 Plotly 没有太大的关系，因此读者可以通过如图 7-8 所示的方式自行研究。

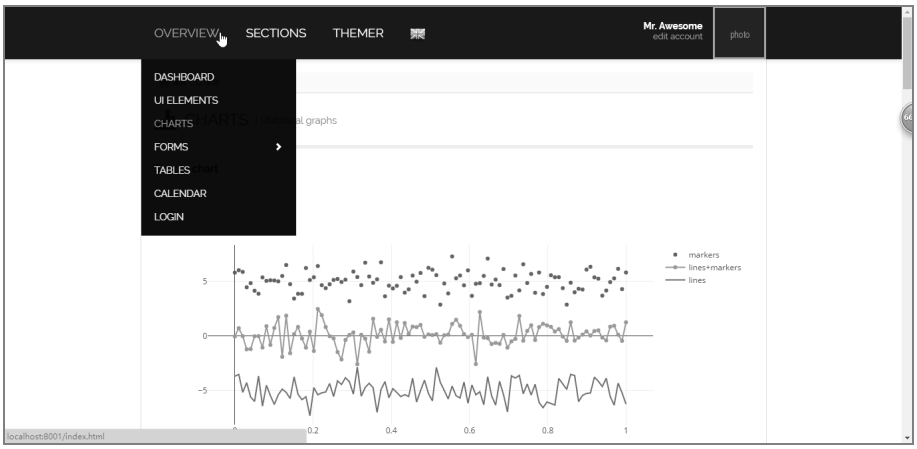

图 7-8　查看其他页面

至此，Plotly 嵌入到 Django 中的案例就介绍完了。本章的重点是提供解决实际应用问题的关键案例，基本上可以满足我们在 Web 开发中的业务需求，对于实际使用中的其他细节（如模板设计、权限管理、连接数据库等），需要读者在实际应用中进一步探索，希望本节内容对读者有所帮助。

7.2　Plotly在Flask中的应用

本节介绍 Plotly 在 Flask 微服务框架中的应用。Flask 是一个非常优秀的 Web 应用框架，有着众多的拥护者，它文档齐全、社区活跃度高、功能强大。本节的内容建立在官方文档（http://docs.jinkan.org/docs/flask/quickstart.html）的基础上，再加上 Plotly 开发，从框架的建立开始，由浅入深地讲解 Flask 加载 Plotly 离线文件。

Flask 是一个使用 Python 语言编写的免费的轻量级 Web 应用框架，其 WSGI 工具箱采用 Werkzeug，模板引擎则使用 Jinja2，如图 7-9 所示。Flask 也被称为“microframework”，因为它使用简单的核心，用 extension 增加其他功能。Flask 没有默认使用的数据库、窗体验证工具。

Flask 处理一个请求的流程是：首先根据 URL 决定由哪个函数来处理，然后在函数中进行操作，取得所需的数据，再将数据传给相应的模板文件，由 Jinja2 负责渲染得到 HTTP 响应内容，最后由 Flask 返回响应内容。

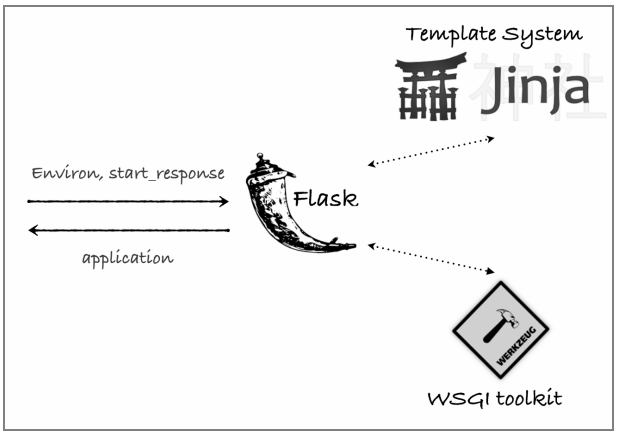

图 7-9　Flask 示意图

Flask 底层使用 werkzeug 来做路由分发，在实际应用中，不同的请求可能会调用相同的处理逻辑。有着相同业务处理逻辑的 HTTP 请求可以用同一类 URL 来标识，比如在论坛站点中，对于所有获取 Topic 内容的请求而言，可以用“topic/<topic_id>/”这类 URL 来表示，这里的 topic_id 用以区分不同的 topic，接着在后台定义一个 get_topic（topic_id）函数，用来获取 topic 相应的数据，此外还需要建立 URL 和函数之间的一一对应关系，这就是 Web 开发中所谓的路由分发，如图 7-10 所示。

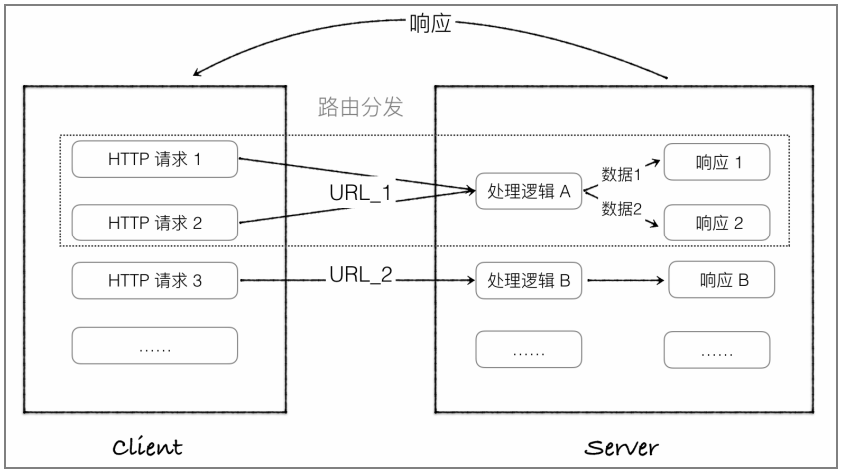

图 7-10　Web 开发中的路由分发

Flask 使用 Jinja2 模板渲染引擎来做模板渲染，通过业务逻辑函数得到数据后，

接下来需要根据这些数据生成 HTTP 响应（对于 Web 应用来说，HTTP 响应一般是一个 HTML 文件）。Web 开发中的一般做法是提供一个 HTML 模板文件，然后将数据传入模板，经过渲染后得到最终需要的 HTML 响应文件。

一种比较常见的场景是，请求虽然不同，但响应中数据的展示方式是相同的。仍以论坛为例，对于不同的主题而言，其具体主题内容虽然不同，但页面展示的方式是一样的，都有标题栏、内容栏等。也就是说，对于主题来说，我们只需提供一个 HTML 模板，然后传入不同的主题数据，即可得到不同的 HTTP 响应，这就是所谓的模板渲染，如图 7-11 所示。

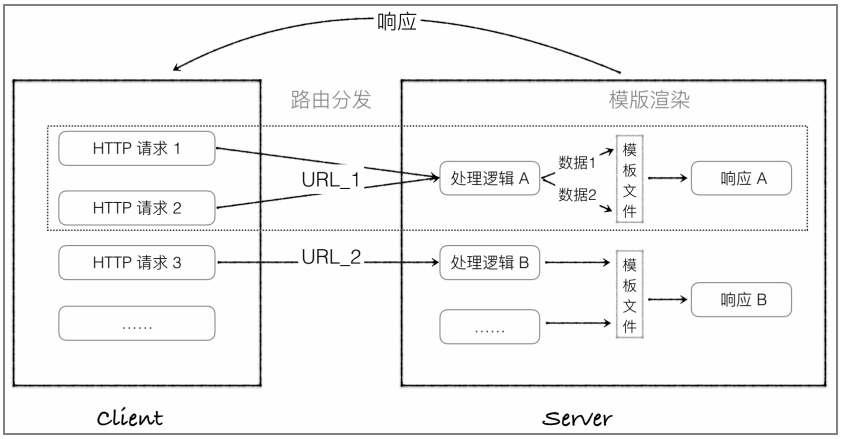

图 7-11　模板渲染

7.2.1　安装 Flask

安装 Flask 的方法特别简单，直接用 pip 命令安装即可：

```
pip install flask
```

7.2.2　最小的 Web 应用

新建项目目录 plotly_and_flask_demo1，在该目录下新建一个 hello.py 文件，具体代码如下。

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

然后运行以下 hello.py 脚本，访问 http://127.0.0.1:5000/，就会看见 Hello World 问候语，如图 7-12 所示。

```
$ python hello.py
 * Running on http://127.0.0.1:5000/
```

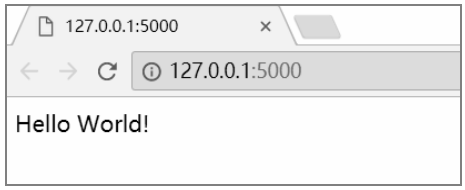

图 7-12　显示 Hello World 问候语

在本例中，使用 route()装饰器告诉 Flask 什么样的 URL 能触发我们的函数。这个函数的名称也在生成 URL 时被特定的函数采用，这个函数返回想要显示在用户浏览器中的信息。

最后用 run()函数让应用运行在本地服务器上，其中“if __name__ == '__main__'”确保服务器只会在该脚本被 Python 解释器直接执行时运行，而不是作为模块导入时运行。

7.2.3　模板渲染

现在的网站很少有全静态页面了，而动态页面靠的就是模板的渲染。Flask 默认的模板引擎是 Jinja2，它有一套自己的模板语言。

在 plotly_and_flask_demo1 目录下新建文件夹 templates，在该文件夹中保存模板

文件。建立一个简单的模板文件 index.html，这个模板文件应该放在主脚本的同目录下的 templates 目录中，具体代码如下。

```
<html>
  <body>
    {% if name %}
      <h1>Hello,{{name}}!</h1>
    {% else %}
      <h1>Hello,Stranger!</h1>
    {% endif %}
  </body>
</html>
```

在 plotly_and_flask_demo1 目录下新建主文件并命名为 template.py，代码如下。

```
from flask import render_template
from flask import Flask

app = Flask(__name__)

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('index.html', name=name)

if __name__ == '__main__':
    app.run()
```

从上面这段代码中我们可以看出以下几点。

（1）路由可以不是固定的，用<变量名>的形式可以制作动态路由。

（2）函数的第一个参数只写了模板的文件名而不是模板的路径，这就是把模板放在固定的 templates 目录下的意思。

Flask 会在 templates 目录里寻找模板。所以，如果用户的应用是一个模块，则这个文件夹应该与模块同级；如果它是一个包，则这个文件夹作为包的子目录。本案例中的脚本结构如下。

```
/template.py
/templates
```

```
    /index.html
```

（3）一个响应函数可以关联多个路由，使这几个路由都定向到这个函数。

在命令行窗口运行 template.py 脚本，代码如下。

```
python template.py
```

然后打开浏览器，在地址栏中输入 http://127.0.0.1:5000/hello，就可以看到网页渲染的模板文件了，如图 7-13 所示。

如果请求的 URL 带参数，则会得到不同的结果，比如访问 http://127.0.0.1:5000/hello/wangwu，会得到如图 7-14 所示的结果。

图 7-13　网页渲染的模板文件

图 7-14　URL 带参数的模板文件

7.2.4　入门案例一

前面介绍了 Flask 的模板渲染，其实很简单，就是在网页中替换特定的变量。本节将讲解在 Flask 框架中使用模板渲染图形，将 Plotly 生成的渲染后的内容以 div 方式嵌入到 HTML 文档中，其实就是在网页中插入一段包含 HTML 的网页，具体步骤如下。

（1）在 plotly_and_flask_demo1 项目目录下新建主脚本 template2.py，代码如下。

```
from flask import render_template
from flask import Flask

import plotly as py
import plotly.graph_objs as go

app = Flask(__name__)
```

```
    @app.route('/')
    def index():
        pyplt = py.offline.plot

        trace0 = go.Bar(
            x=['A 类户型', 'B 类户型', 'C 类户型'],
            y=[20, 14, 23],
            text=['27%市场占有率', '24%市场占有率', '19%市场占有率'],
            marker=dict(
                color='rgb(158,202,225)',
                line=dict(
                    color='rgb(8,48,107)',
                    width=1.5,
                )
            ),
            opacity=0.6
        )

        data = [trace0]
        layout = go.Layout(
            title='2017 年 1 月不同户型房屋单价情况',
        )

        fig = go.Figure(data=data, layout=layout)
        div = pyplt(fig, output_type='div', auto_open=False, show_link=
False)
        context = {}
        context['graph'] = div

        return render_template("index2.html",
            title = 'Home',
            context = context)

    if __name__ == '__main__':
         app.run()
```

以上代码非常简单，就是使用 Plotly 离线生成渲染的内容，然后保存在字典

context 中，最后使用 render_template()函数渲染网页。

（2）建立模板文件 index2.html，保存在 templates 目录下，代码如下。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Plotly 嵌入 Django 案例</title>
    <style type="text/css">
        div.content {
            padding: 1em 5em;
        }
        div.graph {
            margin: 10px auto;
            width: 70%;
        }

    </style>

</head>
<body>
<div class='content'>
    <h2>用 Python 爬虫抓取最新的房价数据</h2>
    <div>
        <p>
        部分城市的房价迅速上涨，多地出台楼市限购政策。未来房价走势如何？由于影响因素的多样性，目前还没有发现任何一种研究分析能够确切地预测，但我们可以自己动手从网络上抓取房价的最新数据，密切关注最新的房价信息，结合实际情况做出自己的判断。
        </p>
        <p>
            下面数据化分析以链家网为例，利用 Python 爬虫抓取三种户型的平均房价（千元/平方米）：
        </p>
    </div>


    <div class='graph'>
```

```
        <h1> {{context.graph |safe }} </h1>

       </div>

        <div>
            <p>从图上可以看出，C 类户型的平均房价最高，最受欢迎，其次是 A 类户型，最后是
B 类户型</p>
        </div>
    </div>

    </body>
    </html>
```

注意，为了让 HTML 正常解析这些标签，需要对 HTML 进行转义，添加 safe 参数即可，代码如下。

```
    <h1> {{context.graph |safe }} </h1>
```

当前项目的目录结构如下：

```
    /template2.py
    /templates
        /index2.html
```

（3）启动本地测试服务器。

在命令行窗口运行 template2.py 脚本，代码如下。

```
    python template2.py
```

然后打开浏览器，在地址栏中输入 http://127.0.0.1:5000，网页中就会插入一段 Plotly 生成的离线渲染内容，如图 7-15 所示。

7.2.5　入门案例二

前面介绍了简单的入门案例，但是这个案例有一个缺点，与“7.1.3 入门案例一”遇到的问题一样，就是传入的 div 参数有点大，约有 2MB。如果一个网页中只有一两个这种绘图，用这个案例的方法是可以接受的，但是如果有多个绘图，就会导致

页面数据量太大而给服务器造成很大的压力，下面来看一下出现这个问题的原因。

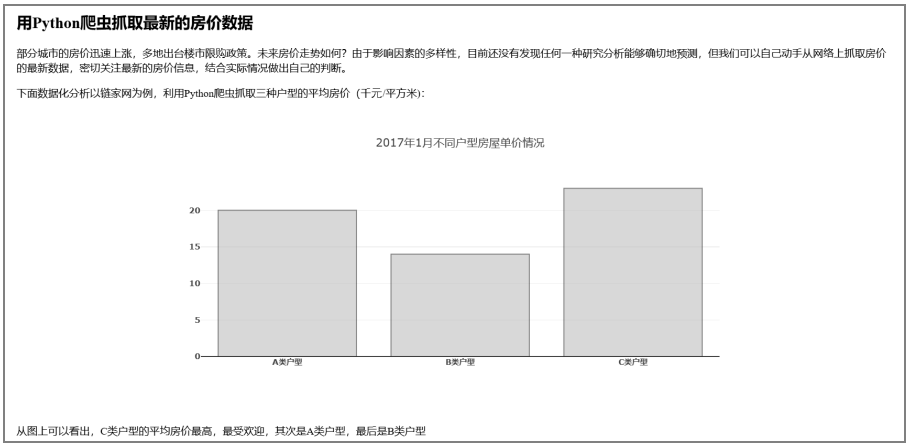

图 7-15　网页中插入的 Plotly 生成的离线渲染内容

注意

这个问题的解决方案与“7.1.4 入门案例二”中所使用的方法一样，考虑到有些读者只对 Flask 熟悉而对 Django 不熟悉，为了照顾这部分读者，在本节中，这个问题的解决方法会与前面的部分内容重复。

首先分析一下我们传递的 div 参数是什么。打开 template2.py 文档并运行，代码如下。

```
from flask import render_template
from flask import Flask

import plotly as py
import plotly.graph_objs as go

app = Flask(__name__)

@app.route('/')
def index():
    pyplt = py.offline.plot

    trace0 = go.Bar(
```

```
            x=['A 类户型', 'B 类户型', 'C 类户型'],
            y=[20, 14, 23],
            text=['27%市场占有率', '24%市场占有率', '19%市场占有率'],
            marker=dict(
                color='rgb(158,202,225)',
                line=dict(
                    color='rgb(8,48,107)',
                    width=1.5,
                )
            ),
            opacity=0.6
        )

        data = [trace0]
        layout = go.Layout(
            title='2017 年 1 月不同户型房屋单价情况',
        )

        fig = go.Figure(data=data, layout=layout)
        div = pyplt(fig, output_type='div', auto_open=False, show_link=
False)
        context = {}
        context['graph'] = div

        import sys
        print('参数 div 占用内存大小为 %d bytes'%sys.getsizeof(div))
        with open('div1.txt', 'w') as file:
            file.write(div)

        return render_template("index2.html",
            title = 'Home',
            context = context)
    if __name__ == '__main__':
        app.run()
```

输出结果如下：

```
参数 div 占用内存大小为 2343823 bytes
```

这个 div 参数非常大，约 2MB，它本质上是 plotly.js 与调用 plotly.js 的绘图代码的结合体，我们打开 div1.txt 就可以看到这个文件前面的内容是 plotly.js 源文件，后面的几行内容才是绘图的代码。我们实际需要的是后面的代码，对于前面的 plotly.js，只要在 HTML 中引用一次就行，没必要每次绘图都重新引用。

打开 div1.txt，会看到如图 7-16 所示的内容。

图 7-16　divl.txt 的内容

图 7-16 中下面的阴影部分才是 Plotly 真正的代码，实际上，我们只要在 HTML 中引用这段代码就行了，plotly.js 可以通过其他方式导入。

下面来看一下如何将剥离 plotly.js 后的 div 嵌入到 Flask 中。这个案例建立在“7.2.4 入门案例一”的基础上，只需要进行一些改动就行了，步骤如下。

（1）修改 template2.py 文件，代码如下。

```
from flask import render_template
from flask import Flask

import plotly as py
```

```
    import plotly.graph_objs as go

    app = Flask(__name__)

    @app.route('/')
    def index():
        pyplt = py.offline.plot

        trace0 = go.Bar(
            x=['A 类户型', 'B 类户型', 'C 类户型'],
            y=[20, 14, 23],
            text=['27%市场占有率', '24%市场占有率', '19%市场占有率'],
            marker=dict(
                color='rgb(158,202,225)',
                line=dict(
                    color='rgb(8,48,107)',
                    width=1.5,
                )
            ),
            opacity=0.6
        )

        data = [trace0]
        layout = go.Layout(
            title='2017 年 1 月不同户型房屋单价情况',
        )

        fig = go.Figure(data=data, layout=layout)
        div = pyplt(fig, output_type='div', include_plotlyjs=False,
auto_open=False, show_link=False)
        context = {}
        context['graph'] = div

        import sys
        print('参数 div 占用内存大小为 %d bytes'%sys.getsizeof(div))
        with open('div1.txt', 'w') as file:
            file.write(div)
```

```
        return render_template("index2.html",
            title = 'Home',
            context = context)

    if __name__ == '__main__':
        app.run()
```

以上代码实现的功能是把 Plotly 的离线绘图结果存储到一个 div 参数（pyplt 返回的结果）里面，这个变量的内容是调用 plotly.js 进行绘图，我们把它放在 context 字典里，传递给 Flask 的 index2.html 模板，最后在 index2.html 模板中显示这个绘图结果。修改的核心代码如下：

```
    div = pyplt(fig, output_type='div', include_plotlyjs=False,
auto_open=False, show_link=False)
```

与之前 Plotly 的绘图函数不同的是，我们这里把结果以.js 的形式存储到一个变量 div 里面（output_type='div'的作用），并设置默认不在网页中打开（auto_open=False 的作用），同时输出结果中不包含 plotly.js（include_plotlyjs=False 的作用）。最后一项 include_plotlyjs=False 是最重要的修改，只有添加了这一项后，我们渲染的 Plotlyjs 图形才不包括 plotly.js 文件。如果没有 include_plotlyjs=False 这一项，Plotly 渲染的图形默认是包括 plotly.js 的，如图 7-17 所示。

offline.py

```
383
384
385 def plot(figure_or_data, show_link=True, link_text='Export to plot.ly',
386          validate=True, output_type='file', include_plotlyjs=True,
387          filename='temp-plot.html', auto_open=True, image=None,
388          image_filename='plot_image', image_width=800, image_height=600,
389          config=None):
```

图 7-17　修改函数代码

（2）修改 templates/index2.html 文件，代码如下。

```
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Plotly 嵌入 Django 案例</title>
```

```
        <style type="text/css">
            div.content {
                padding: 1em 5em;
            }
            div.graph {
                margin: 10px auto;
                width: 70%;
            }

        </style>

        <script  src="{{ url_for('static', filename='js/plotly.min.js') }}"
></script>

    </head>
    <body>
    <div class='content'>
        <h2>用 Python 爬虫抓取最新的房价数据</h2>
        <div>
            <p>
            部分城市的房价迅速上涨，多地出台楼市限购政策。未来房价走势如何？由于影响因
素的多样性，目前还没有发现任何一种研究分析能够确切地预测，但我们可以自己动手从网络上抓取
房价的最新数据，密切关注最新的房价信息，结合实际情况做出自己的判断。
            </p>
            <p>
                下面数据化分析以链家网为例，利用 Python 爬虫抓取三种户型的平均房价（千
元/平方米）:
            </p>
        </div>


        <div class='graph'>

        <h1> {{context.graph |safe }} </h1>

       </div>
```

```
        <div>
            <p>从图上可以看出，C 类户型的平均房价最高，最受欢迎，其次是 A 类户型，最后是
B 类户型</p>
        </div>
    </div>

    </body>
    </html>
```

与之前的 index2.html 文件相比，添加了如下内容：

```
    <script  src="{{ url_for('static', filename='js/plotly.min.js') }}"
></script>
```

我们使用 url_for()函数静态加载/static/js 目录下的 plotly.min.js 文件。使用 Flask 框架只写一行代码就可以使页面动态加载 JavaScript，非常方便，从这一点可以看出 Flask 框架代码的简单和功能强大，Flask 的设计也很人性化，不愧为 Web 开发的“神器”，读者可以将其作为 Web 入门级框架来学习，有兴趣的读者可以深入研究 Flask 框架。

（3）复制 plotly.min.js 文件。

在 plotly_flask_demo1 目录下新建文件夹 static，在 static 目录下新建 js 目录，复制 plotly.min.js 文件（笔者的 Python 环境目录为：d:\Anaconda3\Lib\site-packages\plotly\package_data\plotly.min.js）到 js 目录。项目目录结构如下：

```
    /template2.py
    /templates
        /index2.html
    /static
        /js
            /plotly.min.js
```

（4）启动本地测试服务器，代码如下。

```
    python template2.py
```

打开浏览器，在地址栏中输入 http://127.0.0.1:5000，显示的效果和“入门案例一”的效果是一样的，但是后台输出的结果是：

```
    参数 div 占用内存大小为 804 bytes
```

传入的 div 参数只有 804 字节，大大减轻了服务器加载页面的压力，达到了优化页面的目的，使用这种方式不管我们加载多少个 Plotly 图形，最后在模板页面中加载的都是 Plotly 实际渲染的 HTML 图形。

7.2.6　应用案例

为了便于读者入门，本节介绍相对简单一些的应用案例。与 Plotly 在 Django 中的应用一样，笔者在网上找到了一些适合 Plotly 实战应用的 HTML 模板。由于 Web 开发并不是本章的重点，本章的重点内容是 Plotly 在 Web 开发中的应用，因此本节只介绍 Plotly 嵌入 Web 的相关内容。

我们接下来要做的事情就是在网页上呈现更多的图形信息。新建工程目录并命名为 plotly_flask_demo2，在该目录下新建展示的图形页面，在这里新建两个页面 bar.html 和 charts.html，其他页面还使用原来的模板页面，其他页面不是本节介绍的重点，请读者自行查阅。

在 bar.html 页面添加一个柱状图图形。

在 charts.html 页面添加四个图形（曲线图、柱状图、条形图和饼图）。

plotly_flask_demo2 的项目结构如下：

```
plotly_flask_demo2
├──static
|   ├── css
|   |   └── pages
|   |       └──── dashboard.css
|   |       └──── faq.css
|   |       └──── login.css
|   |       └──── plans.css
|   |   └── adminia.css
|   |   └── adminia-responsive.css
|   |   └── bootstrap.min.css
|   |   └── bootstrap-responsive.min.css
|   |   └── font-awesome.css
|   ├──font
|   |   └── fontawesome-webfont.eot
```

```
    |   |   └── fontawesome-webfont.svg
    |   |   └── fontawesome-webfont.svgz
    |   |   └── fontawesome-webfont.ttf
    |   |   └── fontawesome-webfont.woff
    |   |   └── fontawesome-webfont_162a16fe.eot
    |   ├──img
    |   |   └── body-bg.png
    |   |   └── headshot.png
    |   └──js
    |       └──charts
    |           └──── plotly.min.js
    └──templates
        └── account.html
        └── bar.html
        └── chars.html
        └── faq.html
        └── grid.html
        └── login.html
        └── plans.html
    |
    └──chart_plot.py
    |
    └──plotly_flask.py
```

这个案例的文档结构虽然看上去复杂，但在实际调用中与之前的案例方法是一样的。与之前的内容相比，只是增加了 static 目录，并且模板的 HTML 代码多了一些。各个目录的作用如下。

- /static/css：存放工程需要的页面样式文件。
- /static/font：存放工程需要的页面字体文件。
- /static/img：存放工程需要的图片文件。
- /static/js：存放工程需要加载的 plotly.js 文件。
- /templates：存放工程需要的模板文件。
- chart_plot.py：封装 Plotly 渲染图形的逻辑。
- plotly_flask.py：是 Flask Web 框架的骨架，保存页面的映射路径。

chart_plot.py 封装了 Plotly 渲染图形的业务逻辑，主要有四个图形（曲线图、

柱状图、条形图和饼图），以其中的饼图为例，其代码样例如下。

```
    import numpy as np
    import pandas as pd

    import plotly as py
    import plotly.graph_objs as go

    pyplt = py.offline.plot


    class Chart_Plot:
        def __init__(self):
            pass
        def pie_graph(self):
            ######################################
            #########     饼图        ######
            ######################################

            labels = ['股票', '债券', '现金', '衍生品', '其他']
            values = [33.7, 20.33, 9.9, 8.6, 27.47]
            trace = [go.Pie(labels=labels, values=values)]
            layout = go.Layout(
                title='基金资产配置比例图',
            )
            fig = go.Figure(data=trace, layout=layout)
            div = pyplt(fig, output_type='div', include_plotlyjs=False,
auto_open=False, show_link=False)
            return div
```

plotly_flask.py 针对每个请求，返回相应的 HTML 模板或跳转到指定的 URL，代码如下。

```
    from flask import Flask
    from flask import render_template

    app = Flask(__name__)
    from chart_plot import Chart_Plot
```

```
chart = Chart_Plot()

context = {}

@app.route('/')
def index():
    context['bar_graph'] = chart.bar_graph()
    context['line_graph'] = chart.line_graph()
    context['pie_graph'] = chart.pie_graph()
    context['twoline_graph'] = chart.twoline_graph()

    return render_template("chars.html", title='Home', context=context)

@app.route('/bar')
def index2():
    context['graph'] = chart.bar_graph()

    return render_template("bar.html", title='Home', context=context)

@app.route('/faq')
def faq():
    context = {}

    return render_template("faq.html",title='Home',context=context)

@app.route('/plans')
def plans():
    context = {}

    return render_template("plans.html",title='Home',context=context)

@app.route('/grid')
def grid():
    context = {}

    return render_template("grid.html",title='Home',context=context)
```

```
    @app.route('/charts')
    def charts():
        context = {}

        return render_template("charts.html",title='Home',context=context)

    @app.route('/account')
    def account():
        context = {}

        return render_template("account.html",title='Home',context=
context)

    @app.route('/login')
    def login():
        context = {}

        return render_template("login.html",title='Home',context=context)

    if __name__ == '__main__':
        app.run(port=5000)
```

/templates/chars.html 是整个案例最复杂的一个模板页面，这个模板页面中包含四个 Plotly 图形。加载模板文件用到的样式文件和 plotly.js 文件的核心代码如下。

```
    <link href="{{ url_for('static', filename='css/bootstrap.min.css') }}"
rel="stylesheet" />
        <link href="{{ url_for('static', filename='css/bootstrap-responsive.
min.css') }}" rel="stylesheet" />
        <link href="{{ url_for('static', filename='css/font-awesome.
css') }}" rel="stylesheet" />
        <link href="{{ url_for('static', filename='css/adminia.css') }}"
rel="stylesheet" />
        <link href="{{ url_for('static', filename='css/adminia-responsive.
css') }}" rel="stylesheet" />

        <script src="{{ url_for('static', filename='js/plotly.min.js') }}"
></script>
```

/templates/chars.html 模板页面导航栏的核心代码如下。

```
<ul id="main-nav" class="nav nav-tabs nav-stacked">

    <li >
        <a href="./">
            <i class="icon-home"></i>
            Dashboard
        </a>
    </li>

    <li>
        <a href="/faq">
                <i class="icon-pushpin"></i>
            FAQ
        </a>
    </li>

    <li>
        <a href="/plans">
            <i class="icon-th-list"></i>
            Pricing Plans
        </a>
    </li>

    <li>
        <a href="/grid">
            <i class="icon-th-large"></i>
            Grid Layout
            <span class="label label-warning pull-right">5</span>
        </a>
    </li>

    <li>
        <a href="/account">
            <i class="icon-user"></i>
            User Account
        </a>
```

```
        </li>

        <li>
            <a href="/login">
                <i class="icon-lock"></i>
                Login
            </a>
        </li>

        <li class="active" >
            <a href="/">
                <i class="icon-signal"></i>
                多个图表
            </a>
        </li>

        <li>
            <a href="/bar">
                <i class="icon-user"></i>
                柱状图
            </a>
        </li>

    </ul>
```

/templates/chars.html 模板页面保存 Plotly 图形的核心代码如下。

```
        <div class="widget">
            <div class="widget-header">
                <h3>Area Chart</h3>
            </div> <!-- /widget-header -->

            <div class="widget-content">
                <div class='graph'>
            {{context.twoline_graph |safe }}
                </div>
            </div>
        </div>
```

```
    <div class="widget">
        <div class="widget-header">
            <h3>Line Chart</h3>
        </div> <!-- /widget-header -->
        <div class="widget-content">
            <div class='graph'>
        {{context.line_graph |safe }}
            </div>
        </div>
    </div>

    <div class="widget">
        <div class="widget-header">
            <h3>Bar Chart</h3>
        </div>
        <div class="widget-content">
            <div class='graph'>
        {{context.bar_graph |safe }}
            </div>
        </div>
    </div>

    <div class="widget">
        <div class="widget-header">
            <h3>Pie Chart</h3>
        </div>
        <div class="widget-content">
              <div class='graph'>
              {{context.pie_graph |safe }}
              </div>
        </div>
    </div>
</div>
```

启动本地测试服务器，代码如下。

```
python plotly_flask.py
```

打开浏览器，在地址栏中输入 http://127.0.0.1:5000，显示的效果如图 7-18～图 7-21 所示。

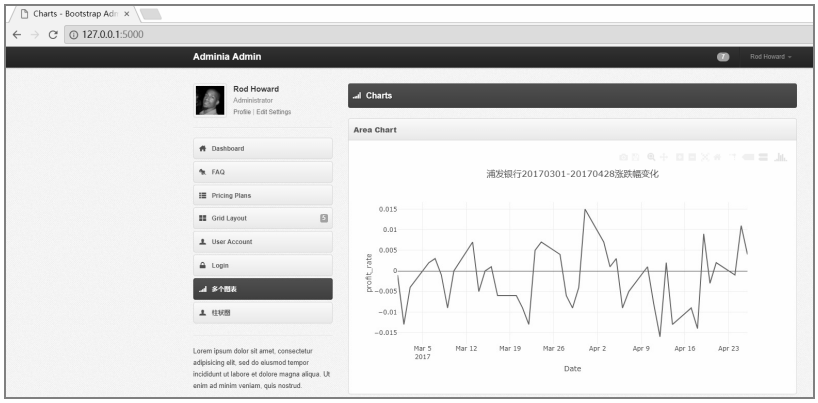

图 7-18　显示效果 1

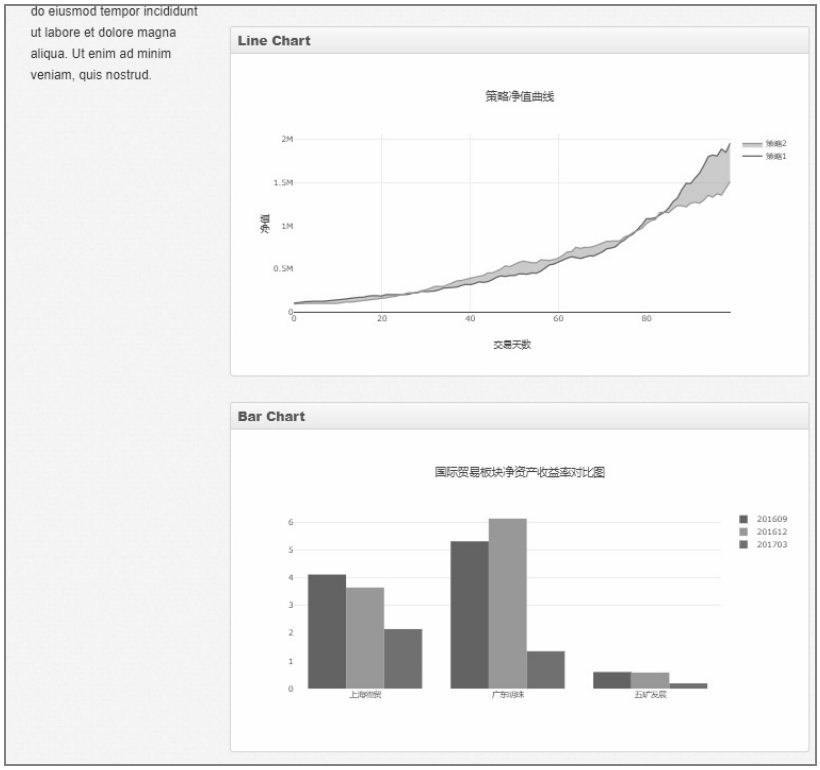

图 7-19　显示效果 2

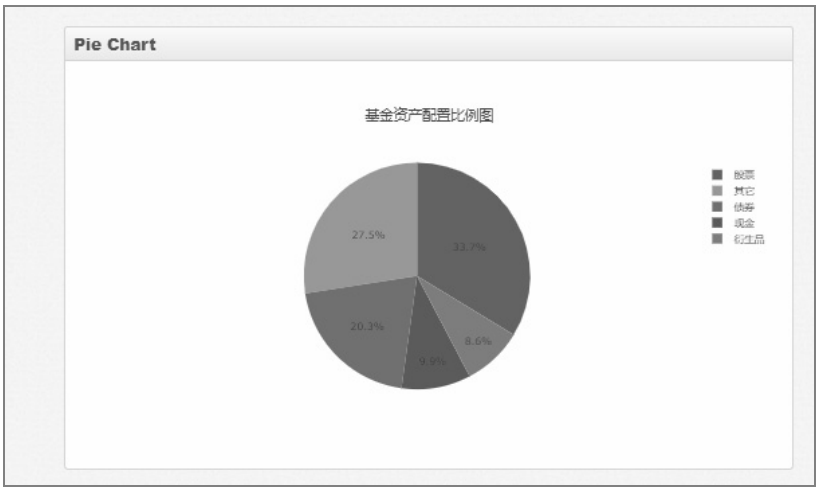

图 7-20　显示效果 3

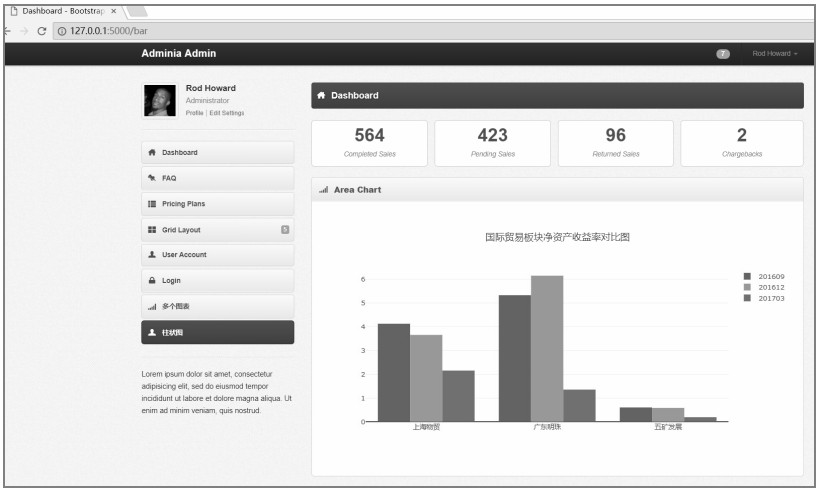

图 7-21　显示效果 4

至此，Plotly 嵌入到 Flask 的案例就介绍完了，从这个应用案例可以看出，Flask 不愧是一个轻量级的 Web 开发框架，用到 Flask 的整个开发文档只有 plotly_flask.py，相比 Django 确实非常精简。本节只提供了 Plotly 嵌入到 Flask 的基本样例，只起到抛砖引玉的作用，对于实际过程中的实战应用还需要读者结合本节的案例不断摸索。

第 8 章

Plotly 与 GUI 开发

Plotly 本质上是基于 JavaScript 的图表库，因此 Plotly 在网页开发方面非常有效。如果想要让 Plotly 嵌入到 GUI 上，那么 GUI 开发框架必须要支持 JavaScript，即一定要有 Web 模块。另外，由于 plotly.js 框架比较新，因此也要求 GUI 开发框架相应的 Web 模块更新速度能够跟得上。最后，从开发者的角度来说，我们希望 GUI 的开发框架使用起来足够简单且好用。

首先，我们需要对 Python 的 GUI 框架有一个基本的了解。从 Python 语言诞生之日起，就有许多优秀的 GUI 框架被整合到 Python 中，使得 Python 也可以在图形界面编程领域大展身手。随着 Python 的流行，许多应用程序都是用 Python 结合这些优秀的 GUI 框架编写的。下面分别介绍 Python GUI 编程的各种实现（内容来自维基百科）。

1．Tkinter

Tkinter 是绑定了 Python 的 Tk GUI 框架，就是 Python 包装的 Tcl 代码，通过内嵌在 Python 解释器内部的 Tcl 解释器实现。将 Tkinter 的调用转换成 Tcl 命令，然后交给 Tcl 解释器进行解释，实现 Python 的 GUI。Tk 和其他语言的绑定，比如 PerlTk，是直接由 Tk 中的 C 库实现的。

Tkinter 是 Python 事实上的标准 GUI，在 Python 中使用 Tk GUI 框架的标准接口，已经包含在 Python Windows 安装程序中，著名的 IDLE 就是使用 Tkinter 实现 GUI 的。

2．wxPython

wxPython 是 Python 对跨平台的 GUI 框架 wxWidgets（用 C++编写）的包装，作为 Python 的一个扩展模块来实现。

wxPython 是比较流行的 Tkinter 的一个替代品，在各种平台上都表现良好。

3．PyGTK

PyGTK 是 Python 对 GTK+GUI 库的一系列包装。

PyGTK 是比较流行的 Tkinter 的一个替代品，GNOME 下许多著名应用程序的 GUI 都是使用 PyGTK 实现的，比如 BitTorrent、GIMP、PyGTK 和 Gedit 都是实现的方式之一。PyGTK 在 Windows 平台上似乎表现不太好，这一点也无可厚非，毕竟使用的是 GTK 的 GUI 库。

4．PySide

PySide 由 Qt 官方维护，目前最新版本是 1.2.2，完成了对 Qt 4.8 版本的完整实现。PySide 是 Python 对跨平台的 GUI 框架 Qt 的另一个包装，捆绑在 Python 中。

PySide 是比较流行的 Tkinter 的一个替代品，拥有 LGPL 2.1 授权许可，允许进行免费的开源软件和私有的商业软件的开发。

在上面的所有图像开发库中，前三个由于没有 Qt Designer（一个 UI 制作工具，它可以通过可视化操作创建 UI 文件，然后通过工具快速编译成.py 文件，因此也可以把它视为一个代码生成器），因此所有的代码都需要手工输入，学习曲线非常的陡峭；而第四个 PySide 本质上也是 Qt 的 Python 封装，只是支持的 Qt 版本比较旧，最新版才支持到 Qt 4.8，而且官方已经停止维护这个库，最近一次更新在 2015 年 10 月 14 日。所以，对于 Python 使用者来说，使用 PyQt 进行 GUI 开发是最好的选择。

另外，从 Plotly 的角度来看，PyQt 可以使用其中的 QWebEngineView 类进行网页开发，并且 QwebEngineView 类使用的是 Chromium 内核，Chromium 内核是 Chrome 浏览器使用的内核，利用 Chrome 浏览器的优势可以完美解决其与 plotly.js 兼容的问题。

综上所述，本章将 PyQt 与 Plotly 结合，详细说明如何把 Plotly 嵌入到 PyQt 里面，但是不会对 PyQt 的基础做过多的介绍，因为这并不是本书的重点，仅仅是 PyQt 的基本使用就足够写一本书了。

在阅读本章之前，读者需要对 PyQt 的基础知识有一些基本的了解。如果对 PyQt 不太感兴趣，也可以略过本章的内容。

8.1　PyQt的安装

由于 QWebEngineView 类目前只支持 PyQt 5 的 PyQt 5.7（包含 5.7）以上版本，所以本章的内容使用 PyQt 5 的最新版即可。PyQt 5 的安装方法很简单，安装命令如下。

```
pip install PyQt5 -i https://pypi.tuna.tsinghua.edu.cn/simple
```

虽然 PyQt 5 的安装比较简单，但是其依赖的一些工具文件如 qtdesigner.exe、uic.exe 等却需要以其他的方式安装，这里推荐使用 Python 的集成开发环境 WinPython（网址是 https://sourceforge.net/projects/winpython/files/，建议选择下载量最大的那个安装包进行下载），里面已经集成了 PyQt 5 所需的所有依赖文件。

注意1

本章的内容建立在读者已经具备了 PyQt 的一些基本知识的基础上，包括能看懂 PyQt 5 最基本的代码，并知道如何运行 PyQt 5 的文件；Qt Designer 的基本使用；把.ui 文件编译成.py 文件等。

如果读者不知道这些知识，也可以试着把本章的程序运行一遍，说不定会有意外的收获。

注意2

本章要求 PyQt 5.7 版本，但是在 Windows 7 下这个版本会导致 Spyder 运行不正常（运行 Spyder 后会出现一个黑色的界面），而在 Windows 8 和 Windows 10 系统中则没有这个问题，因此另一个非常好用的集成开发环境 Anaconda 默认使用的 PyQt 依然是 5.6。如果读者使用的是 Windows 7 系统，建议使用 WinPython 这种便携的开发环境，而不建议升级 Anaconda 的 PyQt 版本，因为那样会导致系统的 Spyder 不能用。

8.2　案例解读

在打算把 Plotly 嵌入到 GUI 开发中之前，笔者一直想要在 Plotly 官网中找到相关的线索，遗憾的是，在 Plotly 的帮助文档中并没有找到与 PyQt 结合使用的具体方

法。经过笔者的实践，发现可以通过 PyQt 的 QWebEngineView 类封装 Plotly 所生成的绘图结果，从而实现 Plotly 与 PyQt 的交互。

这里使用了 QWebEngineView 类，这个类从 PyQt 5.7 版本才开始引入。引入这个类的最主要原因是 PyQt 5.6 及以前版本中使用的是 QWebView 类，QWebView 类使用的是自己开发维护的 WebKit 内核，这个内核比较陈旧，对 JavaScript 的一些新生事物（如 Plotly）的支持性不好。而 QWebEngineView 类使用的是 Chromium 内核，利用 Chrome 浏览器的优势可以完美解决其兼容性问题。但是 QWebEngineView 类有一个比较大的缺点，就是启动速度比较慢，相信在日后的发展中，PyQt 团队会慢慢解决这个问题。

需要提醒读者的是，本案例使用的是 PyQt 5.7 及以上版本，本书 8.6 节会讲解使用 PyQt 5.6 及以前版本与 Plotly 进行间接交互的方法，不过其支持的图表会有限制。

QWebEngineView 类与 Plotly 交互非常简单，本案例文件名为 demo_plotly_pyqt.py，详细代码如下。

```
    # -*- coding: utf-8 -*-

    """
    Module implementing MainWindow.
    """

    from PyQt5.QtCore import *
    from PyQt5.QtGui import *
    from PyQt5.QtWidgets import *
    import sys
    from PyQt5.QtWebEngineWidgets import QWebEngineView


    class Window(QWidget):
    def __init__(self):
           QWidget.__init__(self)
           self.qwebengine = QWebEngineView(self)
    self.qwebengine.setGeometry(QRect(50, 20, 1200, 600))
    self.qwebengine.load(QUrl.fromLocalFile('\plotly_html\if_hs300_bais.h
tml'))
```

```
    app = QApplication(sys.argv)
    screen = Window()
    screen.showMaximized()
    sys.exit(app.exec_())
```

本案例的核心代码如下。

```
    self.qwebengine = QWebEngineView(self)
    self.qwebengine.load(QUrl.fromLocalFile('\plotly_html\if_hs300_bais.h
tml'))
```

以上代码表示新建一个 QWebEngineView 类，以及在 QWebEngineView 类中载入文件。

请注意，if_hs300_bais.html 是用 Plotly 生成的 HTML 本地文件，经过前面章节的学习，读者对利用 Plotly 生成这种文件应该比较熟悉了。我们先看一下程序运行结果，如图 8-1 所示。

图 8-1　程序运行结果

可以看到，这个图非常漂亮，可以动态显示当前时间的价格。在 Plotly 的绘图结果上也可以找到一些其他方法，依次单击右上角的几个按钮就可以使用这些方法。此外，若想查看区间图，可以按住鼠标左键向右拖动，如图 8-2 所示。想要恢复成初始图，单击右上角的“autoscale”按钮即可。

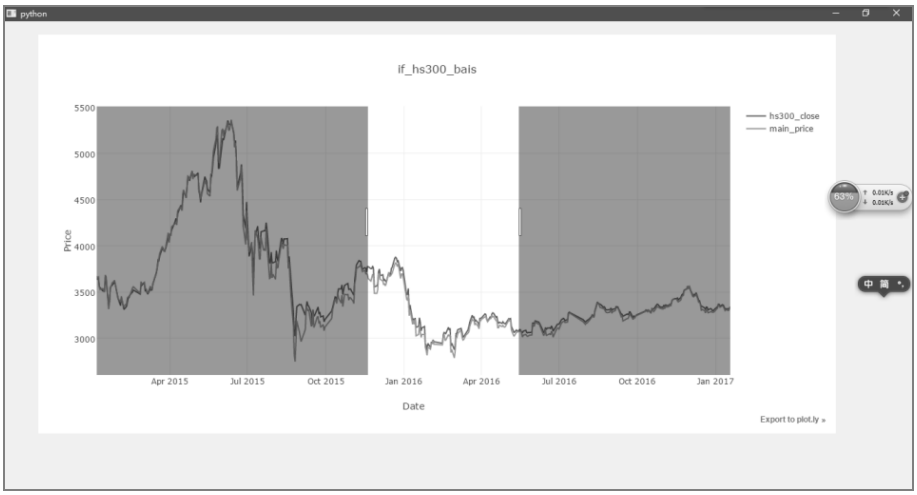

图 8-2　区间图

8.3　设置提升的窗口部件

本节内容属于 PyQt 中 Qt Designer 的一个高级功能，阅读本节内容需要读者已掌握 Qt Designer 的基本用法，如果读者对阅读这节内容感到吃力或对这节内容不感兴趣，也可以略过。之所以要介绍这节内容，是因为这节内容是使用 Qt Designer 把 Plotly 嵌入到 PyQt 中的关键。

所谓提升的窗口部件，是指有些窗口部件是用户自己基于 PyQt 定义的复合窗口部件，这些窗口部件在 Qt Designer 中没有直接提供，但是可以通过提升的窗口部件这个功能来实现，具体方法如下。

从 Container 导航栏中找到 QWidget 并拖入主窗口中，然后对其右击，从弹出的快捷菜单中选择“promote to（提升）”，打开“Promoted Widgets（提升的窗口部件）-Qt Designer”对话框，如图 8-3 所示，按照图中所示进行输入。

单击“Add（添加）”按钮，会发现“Promoted Classes（提升的类）”框中多了一项，如图 8-4 所示。

选中多出来的这一项，然后单击“Promote（提升）”按钮，会在对象查看器中看到如图 8-5 所示的内容，说明已经成功地在 Qt Designer 中引入了 QWebEngineView 类。

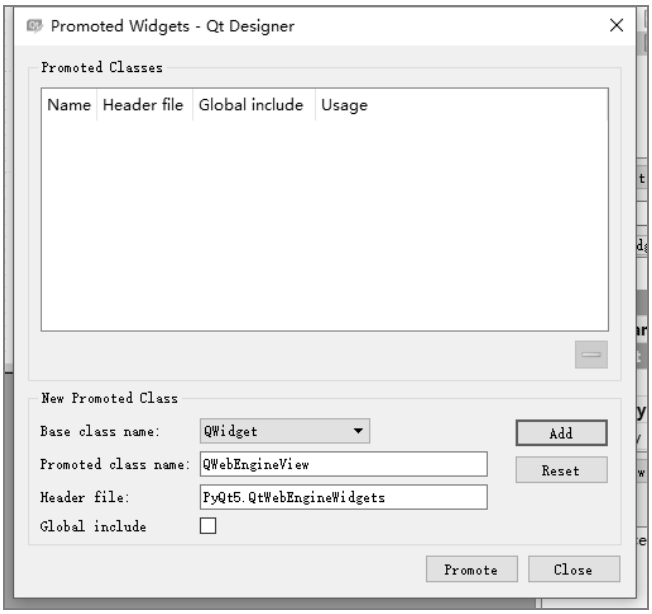

图 8-3　“Promoted Widgets-Qt Designer”对话框

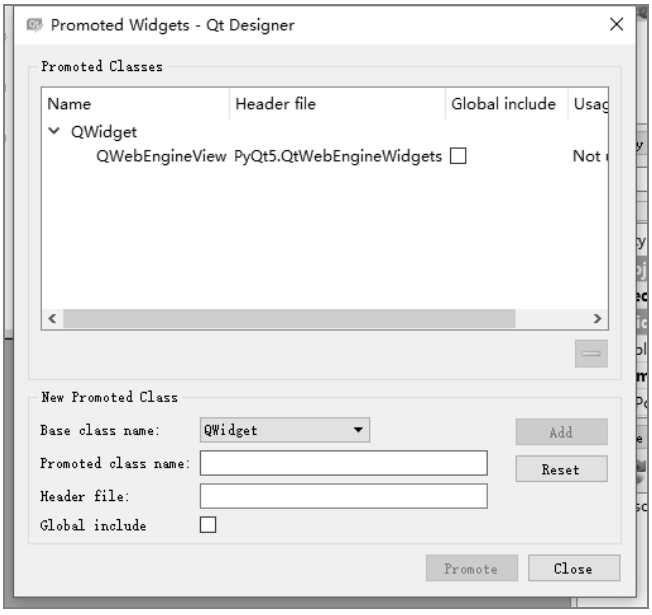

图 8-4　单击“Add”按钮后的“Promoted Widgets-Qt Designer”对话框

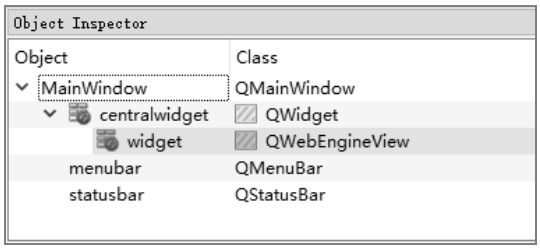

图 8-5　对象查看器

将 widget 重命名为“qwebengine”，这样就基本完成了对提升的窗口部件的操作，代码如下。

```
from PyQt5.QtWebEngineWidgets import QWebEngineView

self.qwebengine = QWebEngineView(self.centralwidget)
self.qwebengine.setGeometry(QtCore.QRect(10, 30, 451, 321))
self.qwebengine.setAutoFillBackground(False)
self.qwebengine.setStyleSheet("background-color: rgb(170, 170, 127);")
self.qwebengine.setObjectName("qwebengine")
```

至此，已经实现了 QWebEngineView 类在 Qt Designer 中的引用。

提升的窗口部件是 PyQt 中非常简单、实用而又强大的功能，利用该功能并通过 Qt Designer 可以实现 PyQt 与 Python 的一些强大的模块之间的交互，可以充分利用 PyQt 和 Python 的优点快速开发程序。

8.4　Plotly_PyQt 5的使用

通过学习前面的案例已知 QWebEngineView 类只需接受 Plotly 生成的 HTML 文件路径就可以实现 PyQt 5 与 Plotly 的交互，因此本案例的主要作用就是通过 Plotly 生成 HTML 本地文件，并返回该 HTML 文件路径（见文件 Plotly_PyQt5.py），代码如下。

```
import pandas as pd
import os
import plotly.offline as pyof
import plotly.graph_objs as go
```

```
    import numpy as np
    import matplotlib.pyplot as plt


    class Plotly_PyQt5():
        def __init__(self):
           '''初始化时设置存储 HTML 文件的文件夹名称，默认为 plotly_html'''
            plotly_dir = 'plotly_html'
            if not os.path.isdir(plotly_dir):
                os.mkdir(plotly_dir)

            self.path_dir_plotly_html = os.getcwd() + os.sep + plotly_dir



        def get_plotly_path_if_hs300_bais(self,file_name='if_hs300_bais.
html'):
            path_plotly = self.path_dir_plotly_html + os.sep + file_name
            df = pd.read_excel(r'if_index_bais.xlsx')


            '''绘制散点图'''
            line_main_price = go.Scatter(
                x=df.index,
                y=df['main_price'],
                name='main_price',
                connectgaps=True, # 这个参数表示允许连接数据缺口
            )


            line_hs300_close = go.Scatter(
                x=df.index,
                y=df['hs300_close'],
                name='hs300_close',
                connectgaps=True,
            )
```

```
        data = [line_hs300_close,line_main_price]

        layout = dict(title='if_hs300_bais',
                      xaxis=dict(title='Date'),
                      yaxis=dict(title='Price'),
                      )

        fig = go.Figure(data=data, layout=layout)

        pyof.plot(fig, filename=path_plotly, auto_open=False)
        return path_plotly
```

对于本案例，需要注意以下几点。

（1）文件绘图使用的是离线模式，而不是在线模式，代码如下。因为离线模式的绘图速度非常快，在线模式绘图由于服务器的原因会比较慢。

```
import plotly.offline as pyof
```

（2）禁止自动在浏览器中打开，需要设置 auto_open 参数为 False，代码如下。

```
pyof.plot(fig, filename=path_plotly, auto_open=False)
```

（3）绘图完成后在本地保存绘图结果，然后在函数里面返回这个路径，最后让 QWebEngineView 类调用这个路径就实现了 PyQt 与 Plotly 的交互，代码如下。

```
return path_plotly
```

对于 pyqt 的主程序，代码如下（见文件 plotly_pyqt.py）。

```
# -*- coding: utf-8 -*-

"""
Module implementing MainWindow.
"""

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMainWindow

from Ui_plotly_pyqt import Ui_MainWindow
```

```
    from PyQt5.QtCore import *
    from PyQt5.QtGui import *
    from PyQt5.QtWidgets import *
    import sys
    from Plotly_PyQt5 import Plotly_PyQt5


    class MainWindow(QMainWindow, Ui_MainWindow):
        """
        Class documentation goes here.
        """

        def __init__(self, parent=None):
            """
            Constructor

            @param parent reference to the parent widget
            @type QWidget
            """
            super(MainWindow, self).__init__(parent)
            self.setupUi(self)
            self.plotly_pyqt5 = Plotly_PyQt5()
            self.qwebengine.setGeometry(QRect(50, 20, 1200, 600))
            self.qwebengine.load(QUrl.fromLocalFile(self.plotly_pyqt5.
get_plotly_path_if_hs300_bais()))


    app = QApplication(sys.argv)
    win = MainWindow()
    win.showMaximized()
    app.exec_()
```

其核心代码如下。

```
    self.plotly_pyqt5 = Plotly_PyQt5()
    self.qwebengine.setGeometry(QRect(50, 20, 1200, 600))
    self.qwebengine.load(QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_
path_if_hs300_bais()))
```

这几行代码的作用类似于前面提到的以下代码：

```
self.qwebengine.load(QUrl.fromLocalFile('\if_hs300_bais.html'))
```

运行结果如图 8-6 所示，与图 8-1 所示的结果没什么不同。

图 8-6　程序运行结果

注意

我们这里使用很笨的方法，首先在 Plotly 中绘图，把结果保存到本地，然后通过 QWebEngineView 类加载这个本地文件，这样就产生了硬盘的写入与读取问题，显然有些拖累程序的运行速度。

针对这个问题，笔者也曾经尝试直接通过 QWebEngineView 类来渲染 Plotly 生成的 JS 代码（见文件 PyQt_plotly_js.py），但遗憾的是没有成功，原因是 QWebEngineView 类对于太大的 JS 代码不能很好地支持，这个属于 PyQt5 的 bug，希望日后这个问题能够完美地解决。所以，本章提供的解决方案虽然不是最完美的方案，但却是目前笔者知道的最好的方案。

8.5　更多扩展（Plotly）

读者如果需要了解更多的案例，可以查阅本书的章节内容，也可以访问官方网

站 https://plot.ly/python/，如图 8-7 所示。

读者需要做的仅仅是对相关案例的代码进行修改，并修改 get_plotly_path_if_hs300_bais 函数，使之支持要实现的绘图结果，就是这么简单！

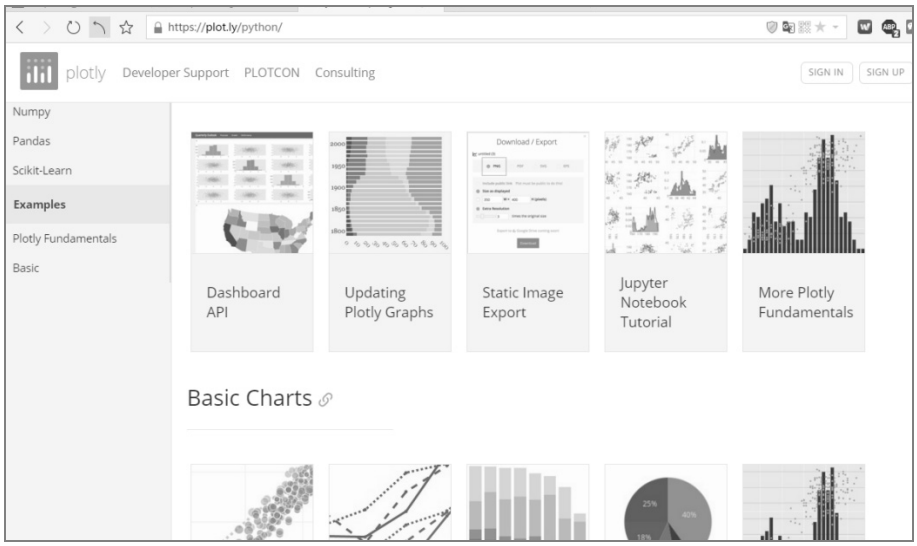

图 8-7　官网案例

8.6　Plotly与PyQt 5.6的结合

由于 Plotly 所采用的 JavaScript 框架比较新，所以 PyQt 5.6 及以前版本对它的支持性不是很好。笔者经过不断摸索，找到了一种 QWebView 支持 Plotly 的间接方案——先用 Matplotlib 进行绘图，然后用 Plotly 渲染结果，最后将其与 QWebView 进行结合。在所有的公开信息中，本书应该是第一个给出这种解决方案的，在这之前还没有见到能够实现 PyQt 5.6 及之前版本与 Plotly 进行交互的案例。

既然这种方案可以解决 PyQt 与 Plotly 进行交互的问题，那么为什么不采用这种方案呢？原因有以下三点。

（1）这种方案严重依赖 Matplotlib 的图表库，如果有些图表在 Matplotlib 中无法实现，那么 Plotly 也就无法对其进行渲染了。也就是说，对于一些 Plotly 拥有而 Matplotlib 中却没有的图表，该方案无能为力。

（2）这种方案所渲染的 Plotly 图表与 Plotly 的原生图表在外观上有一些不同，

Plotly 的原生图表相对好看一些。

（3）有些自定义的 Matplotlib 图表无法通过 Plotly 进行渲染，笔者曾经对双坐标轴的 Matplotlib 图表进行 Plotly 渲染，但是无论如何都没有结果。可见，使用 Plotly 渲染 Matplotlib 图表的兼容性还有待提高。

接下来要说明如何使用 QWebView 与 Plotly 进行交互，幸运的是，Qt Designer 默认为我们提供了 QWebView 类，所以在此就不用设置提升的窗口部件了。使用方法是：在 Qt Designer 界面左侧的“Display Widgets”导航栏中选中 QWebView 并拖动到主窗口中，结果如图 8-8 所示。

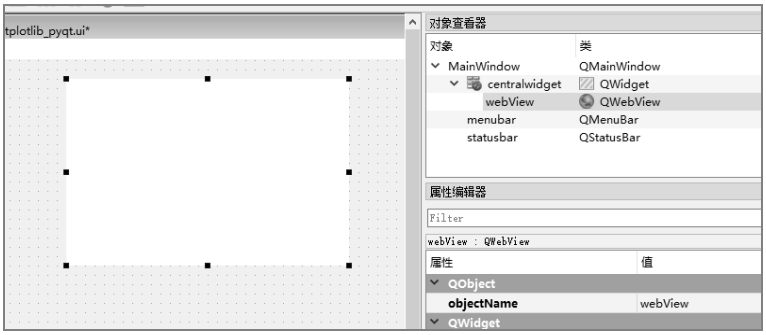

图 8-8　选中 QWebView 并拖到主窗口中

下面给出案例。注意，这个案例需要在 PyQt 5.6 及之前版本中运行，需要读者自行配置相应的 PyQt 环境（见文件 plotly_matplotlib_pyqt.py），代码如下。

```
# -*- coding: utf-8 -*-

"""
Module implementing MainWindow.
"""

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMainWindow

from Ui_plotly_matplotlib_pyqt import Ui_MainWindow

from PyQt5.QtCore import *
from PyQt5.QtGui import *
```

```
    from PyQt5.QtWidgets import *
    import sys
    from Plotly_PyQt5 import Plotly_PyQt5


    class MainWindow(QMainWindow, Ui_MainWindow):
        """
        Class documentation goes here.
        """

        def __init__(self, parent=None):
            """
            Constructor

            @param parent reference to the parent widget
            @type QWidget
            """
            super(MainWindow, self).__init__(parent)
            self.setupUi(self)
            self.plotly_pyqt5 = Plotly_PyQt5()
            self.webView.setGeometry(QRect(50, 20, 1200, 600))
            self.webView.load(QUrl.fromLocalFile(self.plotly_pyqt5.get_
plot_path_matplotlib_plotly()))


    app = QApplication(sys.argv)
    win = MainWindow()
    win.showMaximized()
    app.exec_()
```

其核心代码如下。

```
    self.plotly_pyqt5 = Plotly_PyQt5()
    self.webView.setGeometry(QRect(50, 20, 1200, 600))
    self.webView.load(QUrl.fromLocalFile(self.plotly_pyqt5.get_plot_path_
matplotlib_plotly()))
```

在这里，QWebView 的使用方法和前面介绍的 QWebEngineView 的使用方法一致，把最重要的结果封装到 Plotly_PyQt5 类的 get_plot_path_matplotlib_plotly 函数中。

下面看一下这个函数的使用方法，代码如下。

```
    def get_plot_path_matplotlib_plotly(self,file_name='matplotlib_
plotly.html'):
            path_plotly = self.path_dir_plotly_html + os.sep + file_name

            N = 50
            x = np.random.rand(N)
            y = np.random.rand(N)
            colors = np.random.rand(N)
            area = np.pi * (15 * np.random.rand(N)) ** 2  # 0 to 15 point radii
            scatter_mpl_fig = plt.figure()
            plt.scatter(x, y, s=area, c=colors, alpha=0.5)

            pyof.plot_mpl(scatter_mpl_fig, filename=path_plotly,
resize=True,auto_open=False)
            return path_plotly
```

读者需要注意如下几点。

（1）这个案例使用的是 plot_mpl 函数，而不是 plot 函数。plot_mpl 函数的作用是把 Matplotlib 形式的绘图结果转换成 Plotly 形式的绘图结果。

（2）所传递的第一个参数是 Matplotlib 图表，而不是 Plotly 生成的图表。

（3）resize=True，表示允许 Plotly 重新定义图表的大小，默认为 False。从图 8-9 可以看出，绘图结果的尺寸发生了变化。

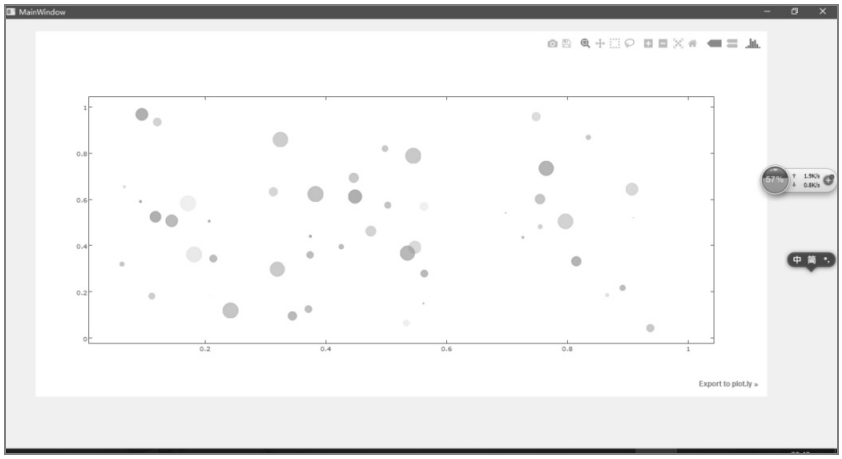

图 8-9　绘图结果

可见，我们实现了 Plotly 对 Matplotlib 图表的渲染并且修改了原有图表的大小（默认的 Matplotlib 图表不会有这么大的宽度）。

8.7　更多扩展（Matplotlib）

读者可以从官方网站（http://matplotlib.org/gallery.html）获取 Matplotlib 的案例（如图 8-10 所示），并修改 get_plot_path_matplotlib_plotly 函数来间接地实现 PyQt（QWebView）与 Plotly 的交互。

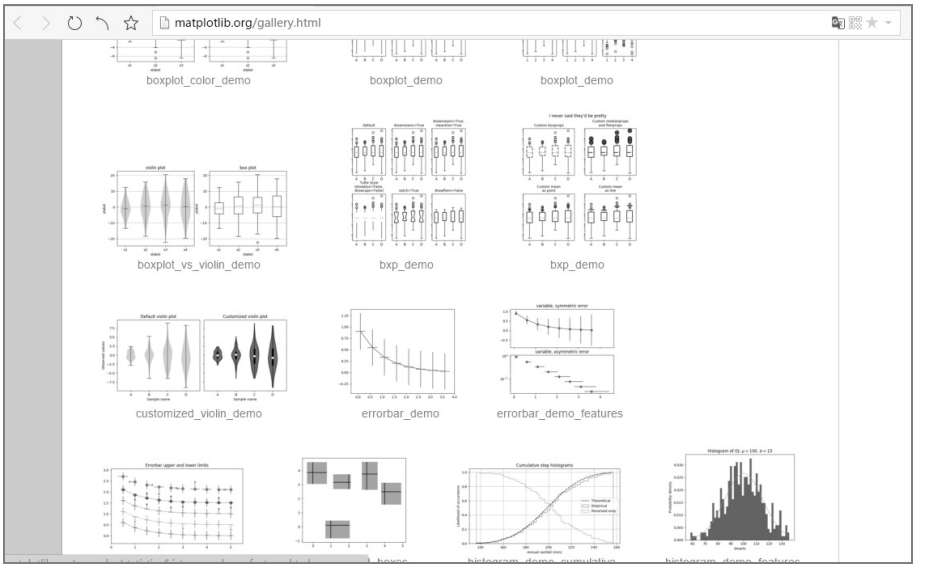

图 8-10　官网案例

8.8　应用案例：展示产品组合信息

本节以金融行业的一个案例来介绍 Plotly 在 PyQt 中的应用。

机构可以从自己的产品信息数据库中筛选出一些优质标的（也就是基金产品），但是对于投资者来说，他们不像机构那么专业，并不能很准确地识别某一种基金产品的好坏，也没有那么多的时间和精力去市场上寻找适合自己投资的基金产品。一边是投资者想要找出适合自己投资的基金产品，另一边是投资者没有能力找出适合

自己投资的基金产品，于是就产生了信息堵塞，其表现就是市场上的一些信息无法有效传递到投资者手中。而机构正好掌握了大量这种信息，具有信息优势，于是机构就可以利用这种信息优势给投资者提供服务，并从中收取服务费。

服务的方式很简单，对于投资者来说，他们其实并不知道适合自己的产品是什么类型的，但是可以预期自己的风险和收益。于是就产生了另一个问题：给定投资者的预期收益与风险的范围，为他们提供最优的投资方案。考虑到风险分散的原则，这个最优的投资计划最好是几种产品的组合。

我们先看一下案例结果，如图 8-11 所示。

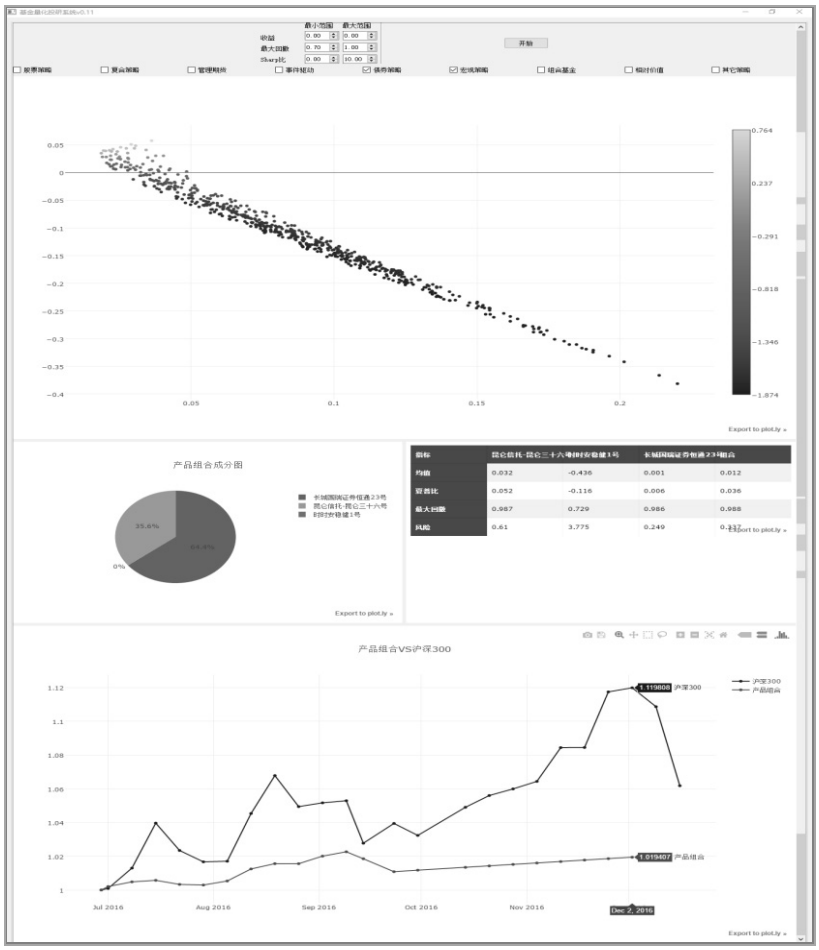

图 8-11　绘图结果

结果分为两部分，上面的部分是让用户选择预期收益、预期风险及偏好的策略类型，如图 8-12 所示。

图 8-12　用户选择部分

下面的部分是从数据库中选取数据，并通过算法找出最优的产品组合及各自的权重，然后用图表展现出来。

提示

出于安全性考虑，这里只提供 GUI 的实现逻辑，而不提供基金产品的数据库，也不提供找出最优产品组合的算法。

对于使用 Qt Designer 进行界面设计这部分内容，这里不再介绍，读者可以参考 combination.ui 文件自行研究。

本案例涉及的代码逻辑并不复杂，当用户输入预期收益和预期风险范围之后，单击“开始”按钮，就会触发唯一的信号槽机制。

```
    @pyqtSlot()
    def on_pushButton_start_combination_clicked(self):
            """
            产品组合分析
            """

            strategy_list = self.check_check_box()
```

self.check_check_box()函数用来检测用户选中的策略是否符合要求，其内容如下，意思是在 strategy_list 中放入已经选中的 checkBox 的名称。

```
    def check_check_box(self):
        strategy_list = []
        for each_checkBox in [self.checkBox_bond,self.checkBox_
combination_fund,self.checkBox_compound,self.checkBox_event,self.checkBo
x_future_manage,self.checkBox_macro,self.checkBox_relative_fund,self.che
```

```
ckBox_others,self.checkBox_others]:
            if each_checkBox.isChecked():
                strategy_list.append(each_checkBox.text())
        return strategy_list
```

下面的内容对于有金融背景的读者来说，理解起来应该没有问题；若没有金融背景，不能理解，则可以略去这部分内容。

当所选择的产品分类数大于 3 和小于 1 时都不合适，会弹出警告框，并视为无效。原因是多个产品分类（大于 3）的组合由于产品之间的多重共线性，会导致分类的模型在数学上没有最优解，即使用计算机暴力算法找出最优解，其权重非零的个数一般也不超过 3 个。所以经过综合考虑，限制产品分类数为 1~3 是最合适的，代码如下。

```
    strategy_list = self.check_check_box()
    if  len(strategy_list) > 3:
        print('最多选择 3 个策略')
        QMessageBox.information(self, "注意", "最多选择 3 个策略")
        return None

    if  len(strategy_list) == 0:
         print('最少选择 1 个策略')
         QMessageBox.information(self, "注意", "最少选择 1 个策略")
         return None
```

接下来设置控件的大小和获取参数信息。由于这里并不准备把参数信息导入数据库中，所以只是简单地输出参数信息，代码如下。

```
    self.QWebEngineview_Combination_monte_markovitz.setMinimumHeight(800)
    self.QWebEngineview_Combination_Pie.setMinimumHeight(400)
    self.QWebEngineview_Combination_Table.setMinimumHeight(400)
    self.QWebEngineview_Combination_Versus.setMinimumHeight(700)

    print('收益_min:', self.doubleSpinBox_returns_min.text())
    print('收益_max:', self.doubleSpinBox_returns_max.text())
    print('最大回撤_min:', self.doubleSpinBox_maxdrawdown_min.text())
    print('最大回撤_max:', self.doubleSpinBox_maxdrawdown_max.text())
    print('sharp 比_min:', self.doubleSpinBox_sharp_min.text())
    print('sharp 比_max:', self.doubleSpinBox_sharp_max.text())
```

现在假设我们已经从数据库中找出组合最优的 3 种产品，分别为昆仑三十六号、时时安稳健 1 号和长城国瑞证券恒通 23 号，它们的组合权重分别为 0.4、0.2 和 0.4，代码如下。

```
    '''假设已经获取产品组合和权重'''
    df = pd.read_excel(r'data\组合.xlsx',index_col=[0])
    w = [0.4, 0.2, 0.4]
    df['组合'] = (df * w).sum(axis=1)
```

最后就是绘图了，代码如下。

```
    self.QWebEngineview_Combination_monte_markovitz.load(
    QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_path_monte_markovitz(
monte_count=600)))
    self.QWebEngineview_Combination_Pie.load(
    QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_path_combination_pie(
df=df, w=w)))
    self.QWebEngineview_Combination_Versus.load(
    QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_path_combination_vers
us(df=df, w=w)))
    self.QWebEngineview_Combination_Table.load(
    QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_path_combination_tabl
e(df=df, w=w)))
```

上面的绘图函数 self.plotly_pyqt5.get_plotly_path_monte_markovitz 对应的图相对有些难度，下面简单介绍一下。

通过建立随机的权重进行蒙特卡洛模拟收益率和方差，然后依次计算出 sharp 比，代码如下。

```
    defget_plotly_path_monte_markovitz(self, file_name='monte_markovitz.
html',monte_count=400,risk_free = 0.03):
        """
        """
        path_plotly = self.path_dir_plotly_html + os.sep + file_name
        df = pd.read_excel(r'data\组合.xlsx',index_col=[0])
        returns = df.pct_change()
        returns.dropna(inplace=True)
        noa = 3
```

```
        # 蒙特卡洛随机模拟结果
        port_returns = []
        port_variance = []

        for p in range(monte_count):
            weights = np.random.random(noa)
            weights /= np.sum(weights)
            port_returns.append(np.sum(returns.mean() * 50 * weights)) # 加入模拟的均值
            port_variance.append(np.sqrt(np.dot(weights.T, np.dot(returns.cov() * 50, weights)))) # 加入模拟的标准差

        port_returns = np.array(port_returns)
        port_variance = np.array(port_variance)
        color_array = (port_returns - risk_free) / port_variance # sharp 比，不同的 sharp 比对应的颜色是不同的
```

接下来对收益率和风险进行绘图，不同的 sharp 比对应不同的颜色，即颜色是可变的，并添加 colorbar，代码如下。

```
    # 此处位置为 get_plotly_path_monte_markovitz 函数内部
    trace1 = go.Scatter(
        x=port_variance,
        y=port_returns,
        mode='markers',
        marker=dict(
           size='6',
            color=color_array,  # 通过一个可变的变量表示颜色，结果是绘图颜色可变
            colorscale='Viridis',
            # 设置 colorbar
            colorbar=dict(
                tickmode='linear',
                tick0=color_array.min(),
                dtick=(color_array.max() - color_array.min()) / 5,
            ),
            showscale=True,
        )
```

```
    )
    data = [trace1]

    pyof.plot(data, filename=path_plotly, auto_open=False)
    return path_plotly
```

打开这个程序，然后运行，结果中使用 QWebEngineView 类所绘制的图表是如此的美丽，这种美丽的动态渲染图在其他 GUI 程序中很少能够见到，而且它的设计又是那么简单。由此，在内心深处对 PyQt 有了特别的喜爱。

第 9 章

Plotly 与机器学习

9.1 Plotly在Sklearn中的应用

本章以讲解案例的方式简单介绍 Plotly 在 Sklearn 中的应用，包括机器学习中的三大问题：分类、回归和聚类结果的可视化。每个部分详细介绍一个案例，重点讲解如何制图，而对于机器学习的算法仅在必要时作简单说明。读者在阅读本章时不必纠结算法层面的问题，Sklearn 是已经封装好的机器学习包，不需要自己编写程序实现的具体算法。

9.1.1 分类问题

本案例使用 SVM 对经典的 IRIS 数据集（鸢尾花）进行分类，选择鸢尾花的两个特征作为自变量，其类别作为因变量，分类可视化结果如图 9-1 所示，代码见文件 plotly_SVM.py。

上述可视化结果的实现包含两部分，一个是绘制热力图，即背景图；另一个是绘制样本点。绘制样本点与本书前面绘制散点图相同，需要注意的是将颜色与分类标签挂钩，即 color=Y，保证同类样本显示同样的颜色，然后使用 colorscale = cmap

给不同的标签填上不同的颜色。

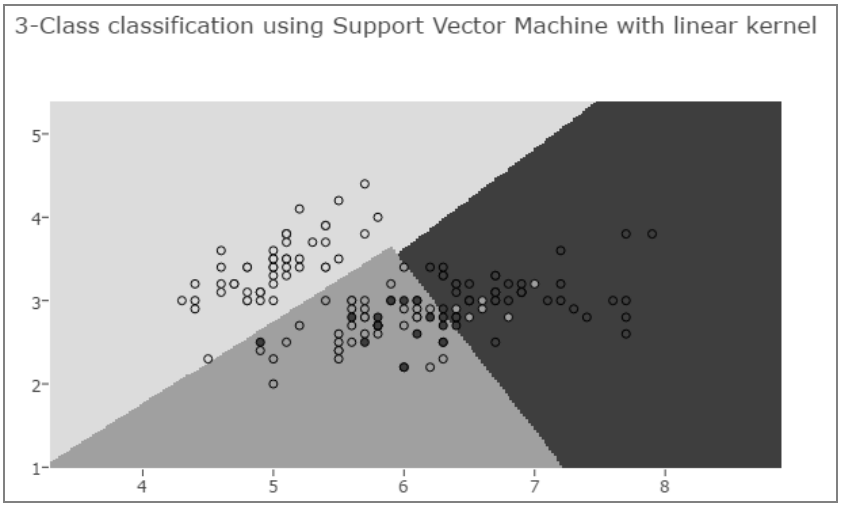

图 9-1　使用线性核函数 SVM 算法三分类结果图

在绘制热力图时，需要使用 plotly.graph_objs 中的 Heatmap 函数，函数中的 *x*、*y*、*z* 三个参数分别对应图像的 *X* 轴坐标、*Y* 轴坐标，以及由 *X* 轴线与 *Y* 轴线相交所得的每个小方块的颜色。如果传递给 *x* 参数的矩阵大小是 a，传递给 *y* 参数的矩阵大小是 b，那么 *z* 参数就需要大小为 a×b 的矩阵，这也是对 *z* 参数做 reshape 操作的原因。案例中传递给 heatmap 函数中参数 *x*、*y* 的是 x_与 y_，均为等差数列，起始点是第一个特征的最小值-1（第二个特征的最小值-1），终点是第一个特征的最大值+1（第二个特征的最大值+1），步长为 0.02；传递给 *z* 参数的则是分类得到的结果矩阵，大小为 220×280。实际上背景图就是由 220×280 个已经着色的小矩阵组成的，由于分类的结果不同，因此呈现的颜色也不同，代码如下。

```
import plotly as py
import plotly.graph_objs as go
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

pyplt = py.offline.plot
iris = datasets.load_iris()
X = iris.data[:, :2] # 只取前两个特征
```

```
    Y = iris.target     # 分类标签
    h = .02  # 设定网格大小
    clf = svm.SVC(kernel='linear')
    clf.fit(X, Y)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    x_ = np.arange(x_min, x_max, h)
    y_ =  np.arange(y_min, y_max, h)
    xx, yy = np.meshgrid(x_, y_)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]) # xx.ravel() 返回连续扁平
的数组

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)

    cmap = [['RGB(10,64,159)'],['RGB(5,159,126)'],['RGB(159,26,64)']]
    # 画布
    trace1 = go.Heatmap(x=x_, y=y_, z=Z,
                    colorscale=cmap,
                    showscale=False)

    # 画点
    trace2 = go.Scatter(x=X[:, 0], y=X[:, 1],
                    mode='markers',
                    marker=dict(color=Y,
                              colorscale=cmap,
                              showscale=False,
                              line=dict(color='black', width=1)))

    layout = go.Layout(title="3-Class classification using Support Vector
Machine with linear kernel")
    fig = go.Figure(data= [trace1, trace2], layout=layout)
    pyplt(fig, filename = r'tmp\SVM_clustering.html')
```

9.1.2　回归问题

回归的可视化相比分类要简单很多，本案例使用三种模型（RBF、Linear、Polynomial）进行回归实践，并可视化结果，数据使用 NumPy 随机生成。上述可视化结果的实现包含两部分，一个是对原始数据点的可视化，另一个是对回归曲线的可视化。在本案例中我们将原始点的 x 与 y 值传递给 p1，将 RBF 拟合的结果传递给 p2，把线性模型拟合的结果传递给 p3，把多项式模型拟合的结果传递给 p4，再将这 4 张图叠加就可以绘制出如图 9-2 所示的回归结果。本案例见文件 plotly_SVR.py，代码如下。

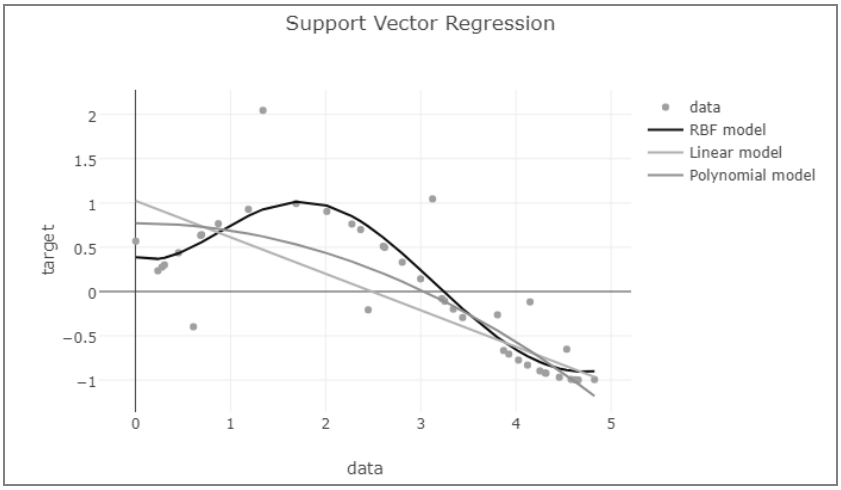

图 9-2　三种算法的回归结果

```
# 3.2 SVR
import plotly as py
import plotly.graph_objs as go
import numpy as np
from sklearn.svm import SVR

pyplt = py.offline.plot

X = np.sort(5 * np.random.rand(40, 1), axis=0)
y = np.sin(X).ravel()
```

```
y[::5] += 3 * (0.5 - np.random.rand(8))

svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_lin = SVR(kernel='linear', C=1e3)
svr_poly = SVR(kernel='poly', C=1e3, degree=2)

y_rbf = svr_rbf.fit(X, y).predict(X)
y_lin = svr_lin.fit(X, y).predict(X)
y_poly = svr_poly.fit(X, y).predict(X)


def data_to_plotly(x):
    k = []
    for i in range(0, len(x)):
        k.append(x[i][0])
    return k

lw = 2
p1 = go.Scatter(x=data_to_plotly(X), y=y,
                mode='markers',
                marker=dict(color='darkorange'),
                name='data')

p2 = go.Scatter(x=data_to_plotly(X), y=y_rbf,
                mode='lines',
                line=dict(color='navy', width=lw),
                name='RBF model')

p3 = go.Scatter(x=data_to_plotly(X), y=y_lin,
                mode='lines',
                line=dict(color='cyan', width=lw),
                name='Linear model')

p4 = go.Scatter(x=data_to_plotly(X), y=y_poly,
                mode='lines',
                line=dict(color='cornflowerblue', width=lw),
                name='Polynomial model')
```

```
layout = go.Layout(title='Support Vector Regression',
                   hovermode='closest',
                   xaxis=dict(title='data'),
                   yaxis=dict(title='target'))

fig = go.Figure(data=[p1, p2, p3, p4], layout=layout)
pyplt(fig, filename = r'tmp\SVR.html')
```

9.1.3　聚类问题

本案例介绍聚类问题的可视化，案例中比较了 K-means 算法和 MiniBatchKmeans 算法的差异，画图部分与 9.1.1 节有相同之处。聚类结果可视化一般分为两部分，一个是画出聚类中心点，另一个是将同一个类别的数据绘制为同一种颜色。这里仍然使用 Scatter 函数完成，详细代码见文件 plotly_Kmeans.py。

本案例代码分为三部分，第一部分产生聚类所需的数据；第二部分分别使用 K-means 算法与 MiniBatchKmeans 算法训练模型；第三部分绘制三张图片，第一张图片是 K-means 分类结果，第二张图片是 MiniBatchKmeans 分类结果，第三张图片突出了两种分类结果的差异。下面详细讲解每个部分。

在第一部分中，需要说明一下 scikit 中的 make_blobs 方法，这种方法常常被用来生成聚类算法的测试数据，会根据用户指定的特征数量、中心点数量、范围等生成几类数据，使得数据可用于测试聚类算法的效果。make_blobs 函数中包含 4 个参数，n_samples 是待生成的样本的总数，n_features 是每个样本的特征数，centers 表示类别数，cluster_std 表示每个类别数据的方差。

在第二部分中，使用生成好的数据调用 K-means 和 MiniBatchKmeans 模型进行聚类，并记录运行时间。

第三部分是绘图部分，首先设定三种类别对应的颜色，然后获取 cluster_centers_ 中的聚类中心点，并使用 pairwise_distances_argmin 函数获取每个数据距离中心点的最短距离。在获取完所需的可视化样本后，使用 make_subplots 生成一个一行三列的画布，并指定标题，这样就可以在绘制完每一幅图后把图片更新到对应的画布位置。在代码中，绘制一幅图后的代码“fig['layout']['xaxis1'].update”就是在做把图片更新

到画布上这件事情。生成画布后，使用 k 循环每次完成一个类别的绘制，kmeans1 中绘制所有同一个类别的样本点，而 kmeans2 中绘制类别的中心点，最后 fig 对应的（1,1）在画布上其实是 6 张图片的叠加，有三张图片依次绘制了三种不同的类别，另外三张图片依次绘制了各个类别的中心点，MiniBatchKmeans 算法结果的可视化与 k-means 算法结果的可视化相同。最后一张图片是对算法结果的差异展示，首先通过逻辑判断找出两种聚类算法结果不同的样本点并在 difference2 中绘制，然后使用 numpy.logical_not 方法选出相同的样本数并在 difference1 中绘制。最后三幅图的绘制结果如图 9-3 所示。

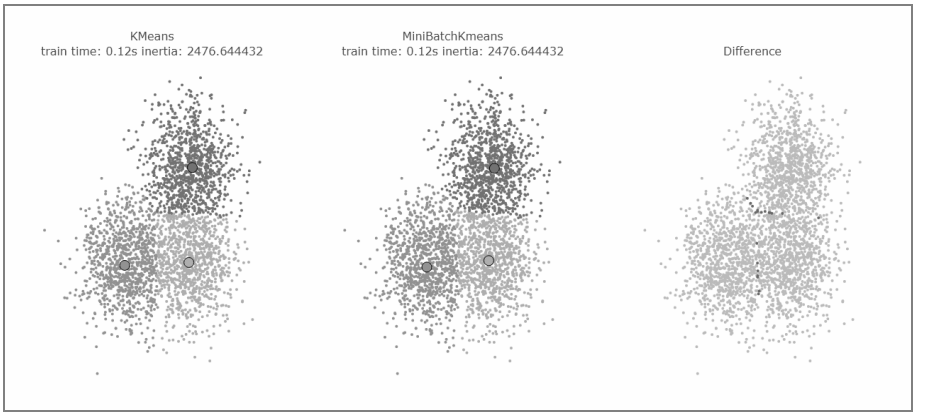

图 9-3　K-means 算法与 MiniBatchKmeans 算法差异比较图

本案例代码如下。

```
import plotly as py
import plotly.graph_objs as go
from plotly import tools

import time
import numpy as np

from sklearn.cluster import MiniBatchKMeans, KMeans
from sklearn.metrics.pairwise import pairwise_distances_argmin
from sklearn.datasets.samples_generator import make_blobs

# 第一部分：产生随机数据
```

```
    np.random.seed(0)

    batch_size = 45
    centers = [[1, 1], [-1, -1], [1, -1]]
    n_clusters = len(centers)
    X, labels_true = make_blobs(n_samples = 3000, centers = centers,
cluster_std = 0.7)

    # 第二部分：模型训练
    # 使用 K-means 算法
    k_means = KMeans(init = 'k-means++', n_clusters = 3, n_init = 10)
    t0 = time.time()
    k_means.fit(X)
    t_batch = time.time() - t0

    # 使用 MiniBatchKmeans 算法
    mbk = MiniBatchKMeans(init = 'k-means++', n_clusters = 3, batch_size =
batch_size,
                          n_init = 10, max_no_improvement = 10, verbose = 0)
    t0 = time.time()
    mbk.fit(X)
    t_mini_batch = time.time() - t0

    # 第三部分
    pyplt = py.offline.plot
    colors = ['#4EACC5', '#FF9C34', '#4E9A06']

    k_means_cluster_centers = np.sort(k_means.cluster_centers_, axis=0)
    mbk_means_cluster_centers = np.sort(mbk.cluster_centers_, axis=0)
    k_means_labels = pairwise_distances_argmin(X, k_means_cluster_centers)
    mbk_means_labels = pairwise_distances_argmin(X, mbk_means_cluster_
centers)
    order = pairwise_distances_argmin(k_means_cluster_centers,
                                      mbk_means_cluster_centers)

    # 第一张图
    fig = tools.make_subplots(rows=1, cols=3,
```

```
                          print_grid=False,
                          subplot_titles=('KMeans<br>train time:
%.2fs\ninertia: %f' %
                                          (t_mini_batch, mbk.inertia_),
                                          'MiniBatchKmeans<br>train time:
%.2fs\ninertia: %f' %
                                          (t_mini_batch, mbk.inertia_),
                                          'Difference'))
    for k, col in zip(range(n_clusters), colors):
        my_members = k_means_labels == k
        cluster_center = k_means_cluster_centers[k]
        kmeans1 = go.Scatter(x=X[my_members, 0], y=X[my_members, 1],
                             showlegend=False,
                             mode='markers', marker=dict(color=col, size=4))
        kmeans2 = go.Scatter(x=[cluster_center[0]], y=[cluster_center[1]],
                             showlegend=False,
                             mode='markers', marker=dict(color=col, size=14,
                                                 line=dict(color='black',
                                                         width=1)))
        fig.append_trace(kmeans1, 1, 1)
        fig.append_trace(kmeans2, 1, 1)

    fig['layout']['xaxis1'].update(showticklabels=False, ticks='',
                                   zeroline=False, showgrid=False)
    fig['layout']['yaxis1'].update(showticklabels=False, ticks='',
                                   zeroline=False, showgrid=False)
    # 第二张图
    for k, col in zip(range(n_clusters), colors):
        my_members = mbk_means_labels == order[k]
        cluster_center = mbk_means_cluster_centers[order[k]]
        minibatchkmeans1 = go.Scatter(x=X[my_members, 0], y=X[my_members,
1],
                                      showlegend=False,
                                      mode='markers', marker=dict(color=col,
size=4))
        minibatchkmeans2 = go.Scatter(x=[cluster_center[0]],
y=[cluster_center[1]],
```

```
                                showlegend=False,
                                mode='markers', marker=dict(color=col,
size=14,

line=dict(color='black',
                                                                width=1)))
        fig.append_trace(minibatchkmeans1, 1, 2)
        fig.append_trace(minibatchkmeans2, 1, 2)

    fig['layout']['xaxis2'].update(showticklabels=False, ticks='',
                               zeroline=False, showgrid=False)
    fig['layout']['yaxis2'].update(showticklabels=False, ticks='',
                               zeroline=False, showgrid=False)

    # 第三张图
    different = (mbk_means_labels == 4)

    for k in range(n_clusters):
        different += ((k_means_labels == k) != (mbk_means_labels == order[k]))

    identic = np.logical_not(different)
    difference1 = go.Scatter(x=X[identic, 0], y=X[identic, 1],
                          showlegend=False,
                          mode='markers', marker=dict(color='#bbbbbb',
size=4))

    difference2 = go.Scatter(x=X[different, 0], y=X[different, 1],
                          showlegend=False,
                          mode='markers', marker=dict(color='magenta',
size=4))

    fig.append_trace(difference1, 1, 3)
    fig.append_trace(difference2, 1, 3)

    fig['layout']['xaxis3'].update(showticklabels=False, ticks='',
                               zeroline=False, showgrid=False)
    fig['layout']['yaxis3'].update(showticklabels=False, ticks='',
```

```
                                zeroline=False, showgrid=False)
pyplt(fig, filename =r tmp\'K-means.html')
```

9.2 PyTorch可视化工具

9.2.1 Visdom 简介

Facebook 的 FAIR 项目组发布了 Visdom，这是一款可以在 Torch、PyTorch 和 NumPy 上实现交互式数据可视化的工具套件。FAIR 项目组表示，他们已经在内部使用了一段时间，认为这款工具非常有助于研究。Visdom 创作者为 FAIR 项目组的 Allan Jabri、Zeming Lin 和 Laurens van der Maaten。其中，Allan Jabri 承担了绝大部分研发工作。

Visdom 的绘图底层使用的是 Plotly 的 JS 接口，所以它可以使用 Plotly 创建基本的图表。Visdom 支持使用 Torch 和 NumPy。Visdom 可视化工具如图 9-4 所示。

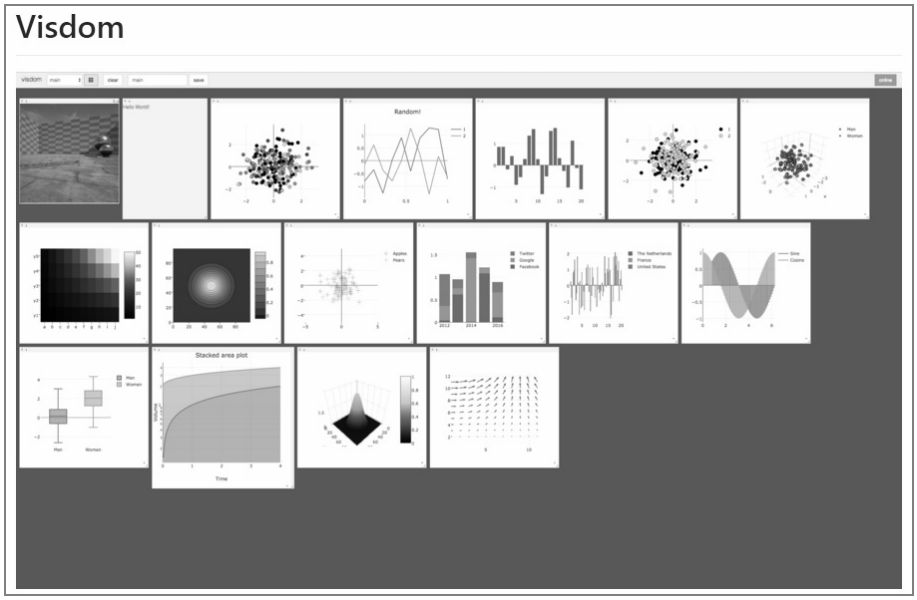

图 9-4　Visdom 可视化工具

Visdom 是一个创造可视化数据的工具，它已经被 Facebook 开源了，感兴趣的读者可以访问 Visdom 的 GitHub 网址 https://github.com/facebookresearch/visdom。

9.2.2　安装 Visdom

在 Python 3 环境下，使用 pip 命令安装 Visdom 非常简单，直接用 pip 命令安装即可，代码如下。

```
pip install visdom
```

启动服务器，代码如下。

```
python -m visdom.server
```

一旦启动服务器，就可以通过在浏览器的地址栏中输入 http://localhost:8097 来访问 Visdom，localhost 可以换成 Visdom 的云主机地址，如图 9-5 所示。

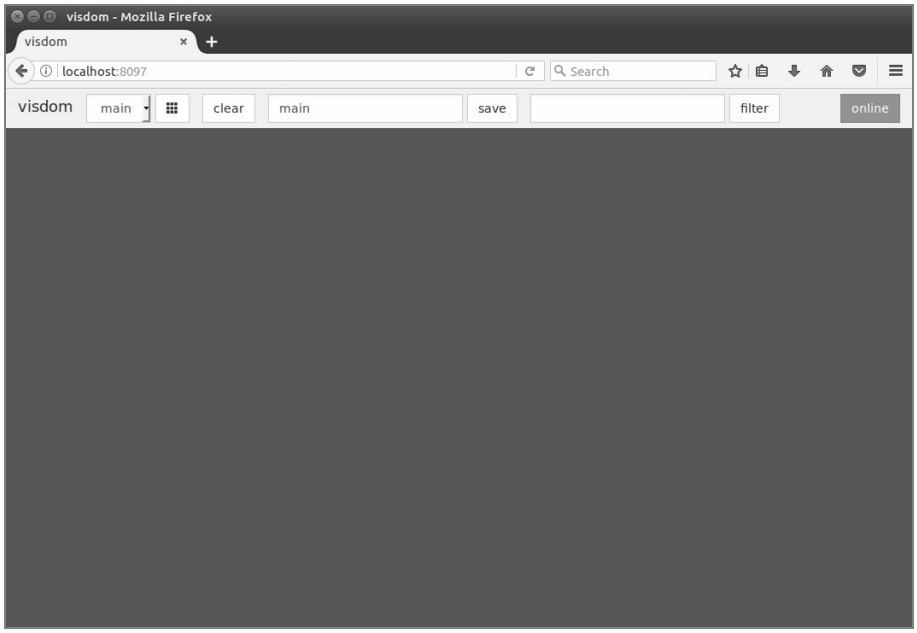

图 9-5　访问 Visdom

如果环境配置成功，则可以看到一个空白的窗格，当前的会话环境为 main。

9.2.3 Visdom 与 Plotly

Visdom 底层虽然用的是 Plotly.js，但是与 Plotly 的呈现方式却不同：Plotly 的绘图结果是一个绘图使用一个网页，而 Visdom 则是把所有绘图结果封装在一个统一的界面去管理，并且可以任意调整每个绘图的大小与位置，这样就可以很方面地管理历史绘图信息，比如可以删掉一些无用的绘图结果，只保留有效的绘图结果。这一点在机器学习中具有比较重要的意义，通过这种方式我们可以查看同一种训练方法传递不同的参数后所得绘图结果的前后对比，这也是 PyTorch 使用 Visdom 进行绘图的原因。

9.2.4 Visdom 基本概念

Visdom 用 JavaScript 在网页上实现了类似 MATLAB 和 Python Matplotlib 的图形展示功能，支持数十种图形，包括 2D 和 3D 图形，交互很流畅，足以满足一般科学计算的需要。

1. Panes（窗格）

Panes 就是用于绘图的小窗格，在代码中叫 Window，可以使用浏览器的放大和缩小功能来调整用户界面的大小，如图 9-6 所示。

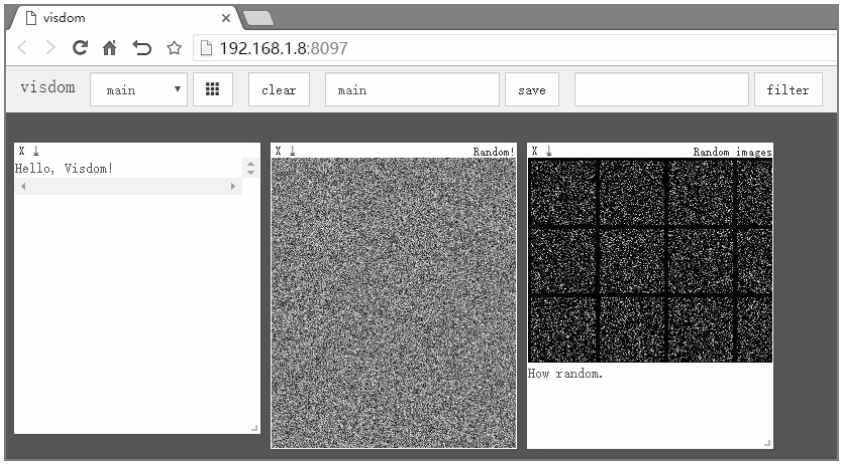

图 9-6 Panes（窗格）

用户界面刚开始是一个白板，用户可以用图像和文本填充它。这些填充的数据出现在 Panes 中，用户可以对这些 Panes 进行拖放、删除、调整大小和销毁操作。Panes 是保存在 Environments（环境）中的，Environments 的状态存储在会话之间。用户可以下载 Panes 中的内容，包括在 SVG 中的绘图。

2．Environments（环境）

可以使用 Environments 对可视化空间进行分区。默认情况下，每个用户都有一个叫作 main 的 Environments。可以通过编程或用户界面创建新的 Environments。Environments 的状态是长期保存的。

可以通过 http://localhost.com:8097/env/main 访问本地的 Environments。如果用户的 Visdom 服务器是部署在远程服务器上的，那么可以将此 URL 分享给其他人，其他人也能看到该用户的可视化结果。

在初始化服务器时，用户的 Environments 默认通过“$HOME/.visdom/”加载，也可以将自定义的路径当作命令行参数传入。如果移除了 Env 文件夹下的.json 文件，那么相应的环境也会被删除。

3．State（状态）

一旦创建了一些可视化，状态是被保存的。服务器自动缓存用户的可视化——如果重新加载网页，则用户的可视化会重新出现。

- Save：通过单击 save 按钮，用户可以手动保存 Environments。保存时首先序列化 Env 的状态，然后以.json 文件的形式保存到硬盘上，包括窗口的位置。同样，用户也可以通过编程来实现 Env 的保存。当面对一些十分复杂的可视化时，参数的设置非常重要，这种保存 Env 状态的方法是十分有用的。例如，数据丰富的演示、模型的训练和系统实验，这些设计依旧可以使这些可视化十分容易分享和复用。
- Fork：如果用户输入了一个新的 Environments 名字，保存时就会建立一个新的 Environments，有效地复制（forking，等价于 GitHub 的 fork）之前的状态。

9.2.5　Visdom 经典案例

Visdom 支持下列 API。由 Plotly 提供可视化支持。

- vis.text：文本。
- vis.image：图片。
- vis.scatter：2D 或 3D 散点图。
- vis.line：线图。
- vis.stem：茎叶图。
- vis.heatmap：热力图。
- vis.bar：条形图。
- vis.histogram：直方图。
- vis.boxplot：箱型图。
- vis.surf：表面图。
- vis.contour：轮廓图。
- vis.mesh：网格图。
- vis.svg：SVG 图像。

这些 API 的确切输入类型有所不同，尽管大多数 API 的输入包含一个 tensor X（保存数据）和一个可选的 tensor Y（保存标签或时间戳）。所有的绘图函数都接收一个可选参数 win，用来将图画到一个特定的 Window（窗格）上。每个绘图函数也会返回当前绘图的 win 参数。用户也可以指定绘出的图添加到哪个 Environments 上（这里 Window 的意思就是之前讲的 Panes）。

1．查看 Visdom 模块的帮助实例

可以使用 Python 的原生命令 help 来查看 Visdom 模块下函数的参数与属性，使用的代码如下。

```
from visdom import Visdom

vis= Visdom()
print(help(vis.text ) )
```

Visdom 其他绘图的内置函数也可以使用这种方法查看有关函数的内置属性，输出信息如下。

```
Help on method text in module visdom:

text(text, win=None, env=None, opts=None) method of visdom.Visdom
```

```
instance
        This function prints text in a box. It takes as input an `text` string.
        No specific `opts` are currently supported.
```

2．vis.text

使用此函数可在文本框中打印文本。

本案例文件名为 Chapter09/pt09_vm01_text.py，在文本框内输入一个 text 字符串，代码如下。

```
from visdom import Visdom

vis= Visdom()
vis.text('Hello, world !')
```

启动 Visdom 服务器后，运行脚本，访问 http://localhost:8097 的显示效果如图 9-7 所示。

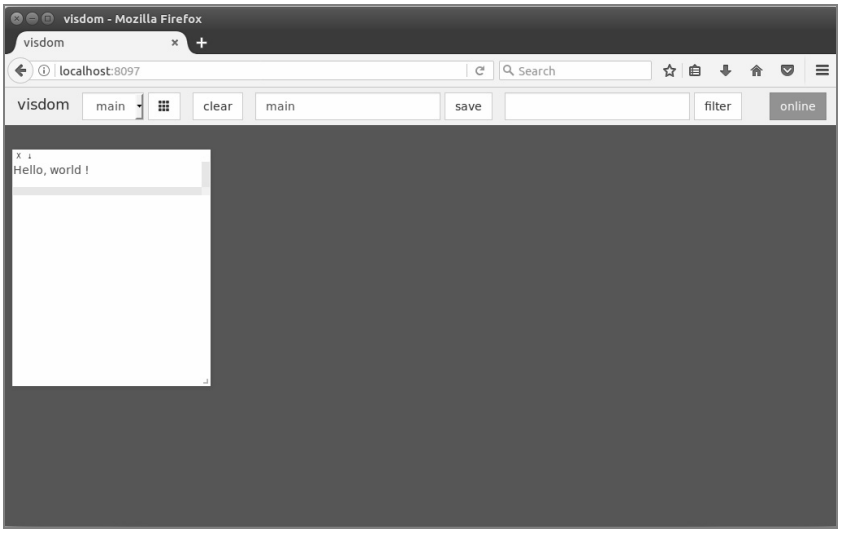

图 9-7　显示文本

3．vis.image

vis.image 函数用来显示图片。本案例文件名为 Chapter09/pt09_vm02_img.py，演示在页面中显示图片，代码如下。

```
from visdom import Visdom

vis= Visdom()

# 显示单图片
vis.image(
    np.random.rand(3, 256, 256),
    opts=dict(title='单图片', caption='图片标题 1'),
)

# 显示网格图片、多张图片
vis.images(
    np.random.randn(20, 3, 64, 64),
    opts=dict(title='网格图像', caption='图片标题 2')
)
```

运行脚本，显示效果如图 9-8 所示。

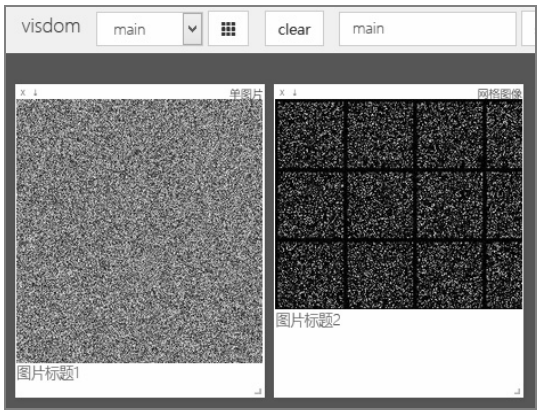

图 9-8　显示图片

4．vis.scatter

vis.scatter 函数用来画 2D 和 3D 数据的散点图。它需要输入 *N*×2 或 *N*×3 的 tensor X 来指定 *N* 个点的位置。一个可供选择的长度为 *N* 的 vector 用来保存 X 中的点对应的标签（1～K），标签可以通过点的颜色反映出来。

本案例文件名为 Chapter09/pt09_vm03_scatter.py，代码如下。

```
from visdom import Visdom
import numpy as np

vis = Visdom()

# 2D scatterplot with custom intensities (red channel)
vis.scatter(
    X =  np.random.rand(255, 2),
    Y = (np.random.randn(255) > 0) + 1 ,
    opts=dict(
        markersize=10,
        markercolor=np.floor(np.random.random((2, 3)) * 255),
    legend=['Men', 'Women']
    ),
)
```

运行脚本，显示效果如图 9-9 所示。

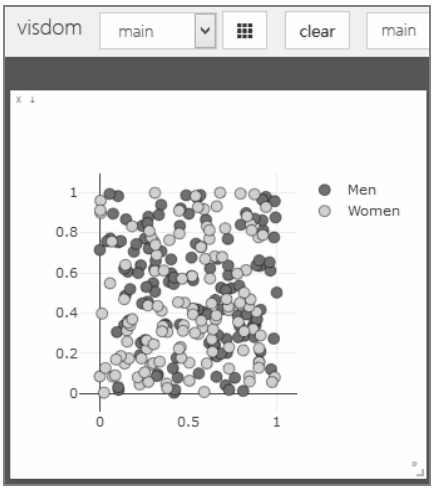

图 9-9　散点图

5．vis.line

vis.line 函数用来画线形图。该函数需要一个形状为 N 或 $N\times M$ 的 tensor Y，用来指定 M 条线的值（每条线上有 N 个点），以及一个可供选择的 tensor X，用来指定对应的 X 轴的值，X 可以是一个长度为 N 的 tensor（这种情况下，M 条线共享同一

个 *X* 轴），也可以是形状和 Y 一样的 tensor。

该函数支持以下属性。

- options.fillarea：填充线下面的区域（boolean）。
- options.colormap：颜色图（string; default = \'Viridis\'）。
- options.markers：显示点标记（boolean; default = false）。
- options.markersymbol：标记的形状（string; default = \'dot\'）。
- options.markersize：标记的大小（number; default = \'10\'）。
- options.legend：包含图例名称的表。

本案例文件名为 Chapter09/pt09_vm04_line.py，代码如下。

```
from visdom import Visdom
import numpy as np

vis = Visdom()

# line plots
Y = np.linspace(-5, 5, 100)
vis.line(
    Y=np.column_stack((Y * Y, np.sqrt(Y + 5))),
    X=np.column_stack((Y, Y)),
    opts=dict(markers=False),
)
```

运行脚本，显示效果如图 9-10 所示。

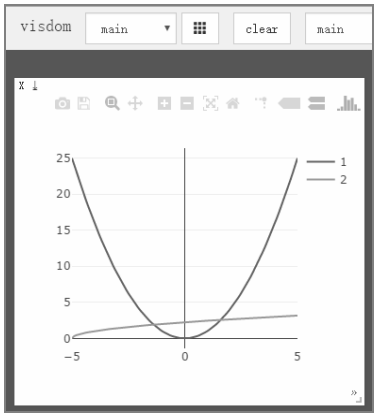

图 9-10　线形图

6．vis.stem

vis.stem 函数用来画茎叶图。该函数需要一个形状为 N 或 $N\times M$ 的 tensor X 来指定 M 时间序列中 N 个点的值，以及一个可选择的 Y，形状为 N 或 $N\times M$，用 Y 来指定时间戳，如果 Y 的形状是 N，那么默认 M 时间序列共享同一个时间戳。

该函数支持以下特定选项。

- options.colormap：颜色图（string; default = \'Viridis\'）。
- options.legend：包含图例名称的表。

本案例文件名为 Chapter09/pt09_vm05_stem.py，代码如下。

```
from visdom import Visdom
import numpy as np
import math

vis = Visdom()

# stemplot
Y = np.linspace(0, 2 * math.pi, 70)
X = np.column_stack((np.sin(Y), np.cos(Y)))
vis.stem(
    X=X,
    Y=Y,
    opts=dict(legend=['Sine', 'Cosine'])
)
```

运行脚本，显示效果如图 9-11 所示。

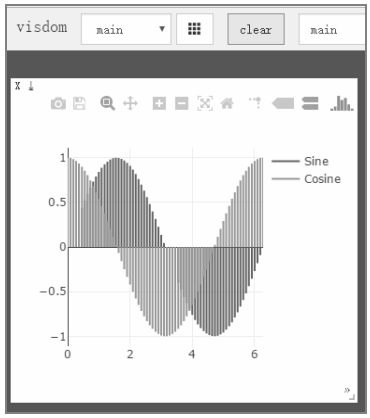

图 9-11　茎叶图

7．vis.heatmap

vis.heatmap 函数用来画热力图。该函数需要输入一个形状为 $N \times M$ 的 tensor X，X 指定了热力图中位置的值。

该函数支持下列特定选项。

- options.colormap：颜色图（string; default = \'Viridis\'）。
- options.xmin：小于这个值时会被剪切成这个值（number; default = X:min()）。
- options.xmax：大于这个值时会被剪切成这个值（number; default = X:max()）。
- options.columnnames：包含 X 轴标签的表。
- options.rownames：包含 Y 轴标签的表。

本案例文件名为 Chapter09/pt09_vm06_heatmap.py，代码如下。

```
from visdom import Visdom
import numpy as np
import math

vis = Visdom()

vis.heatmap(
    X=np.outer(np.arange(1, 6), np.arange(1, 11)),
    opts=dict(
        columnnames=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'],
        rownames=['y1', 'y2', 'y3', 'y4', 'y5'],
        colormap='Electric',
    )
)
```

运行脚本，显示效果如图 9-12 所示。

8．vis.bar

vis.bar 函数可以画普通的、堆起来的或分组的条形图。该函数的输入参数如下。

- X（tensor）：形状 N 或 $N \times M$，指定每个条形的高度。如果 X 有 M 列，那么每行的值可以看作一组或把它们的值堆起来（取决于 options.stacked 是否为 True）。
- Y（tensor, optional）：形状 N，指定对应的 X 轴的值。

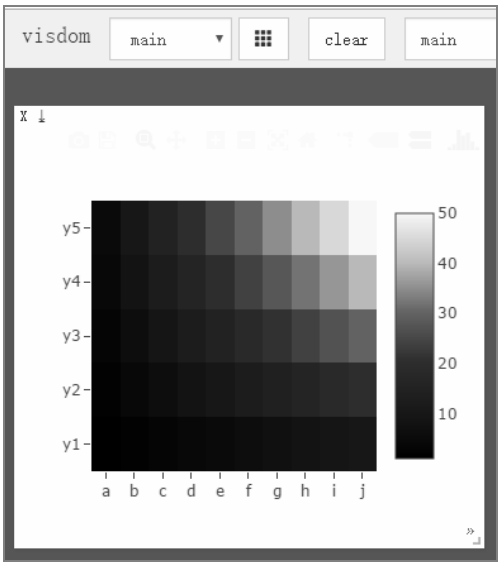

图 9-12　热力图

该函数支持以下特定选项。

- options.columnnames：包含 *X* 轴标签的表格。
- options.stacked：在 X 中堆叠多个列。
- options.legend：包含图例名称的表。

本案例文件名为 Chapter09/pt09_vm07_bar.py，代码如下。

```
from visdom import Visdom
import numpy as np
import math

vis = Visdom()

# 单个条形图
vis.bar(X=np.random.rand(20))

# 堆叠条形图
vis.bar(
    X=np.abs(np.random.rand(5, 3)),
    opts=dict(
        stacked=True,
```

```
        legend=['Sina', '163', 'AliBaBa'],
        rownames=['2013', '2014', '2015', '2016', '2017']
    )
)

# 分组条形图
vis.bar(
    X=np.random.rand(20, 3),
    opts=dict(
        stacked=False,
        legend=['A', 'B', 'C']
    )
)
```

运行脚本，显示效果如图 9-13 所示。

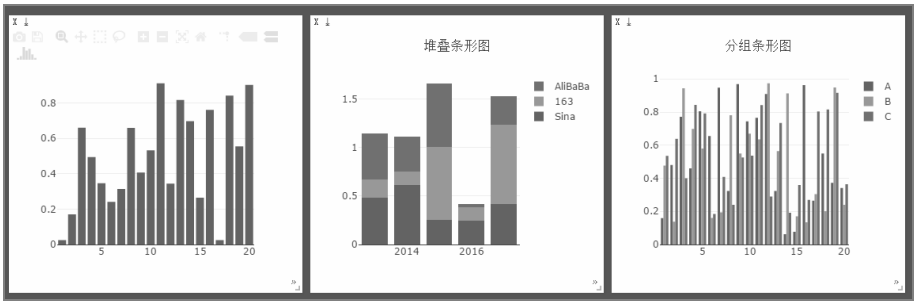

图 9-13　条形图

9．vis.histogram

vis.histogram 函数用来画指定数据的直方图，需要输入长度为 *N* 的 tensor X，X 保存了构建直方图的值。

该函数支持下面的特定选项。

- options.numbins：bins 的个数（number; default = 30）。

本案例文件名为 Chapter09/pt09_vm06_heatmap.py，代码如下。

```
from visdom import Visdom
import numpy as np

vis = Visdom()
```

```
vis.histogram(X=np.random.rand(10000), opts=dict(numbins=20))
```

运行脚本，显示效果如图 9-14 所示。

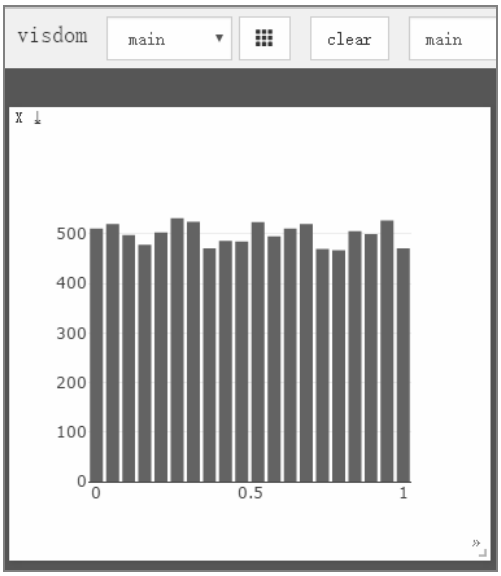

图 9-14　直方图

10．vis.boxplot

vis.boxplot 函数用来画箱型图，其输入参数如下。

- X（tensor）：形状 N 或 $N \times M$，指定做第 M 个箱型图的 N 个值。

该函数支持下面的特定选项。

- options.legend：数据 X 中每一列的标签。

本案例文件名为 Chapter09/pt09_vm09_boxplot.py，代码如下。

```
from visdom import Visdom
import numpy as np

vis = Visdom()

# boxplot
X = np.random.rand(100, 2)
X[:, 1] += 2
```

```
vis.boxplot(
    X=X,
    opts=dict(legend=['Men', 'Women'])
)
```

运行脚本，显示效果如图 9-15 所示。

图 9-15　箱型图

11．vis.surf

vis.surf 函数用来画表面图，其输入参数如下。

- X（tensor）：形状 $N \times M$，指定表面图上位置的值。

该函数支持以下特定选项。

- options.colormap：颜色谱，字符串格式，默认为“Viridis（翠绿色）”。
- options.xmin：设置最小值，数字格式，默认是 X 的最小值。
- options.xmax：设置最大值，数字格式，默认是 X 的最大值。

本案例文件名为 Chapter09/pt09_vm10_surf.py，代码如下。

```
from visdom import Visdom
import numpy as np
```

```
vis = Visdom()

x = np.tile(np.arange(1, 101), (100, 1))
y = x.transpose()
X = np.exp((((x - 50) ** 2) + ((y - 50) ** 2)) / -(20.0 ** 2))

vis.surf(X=X, opts=dict(colormap='Hot'))
```

运行脚本，显示效果如图 9-16 所示。

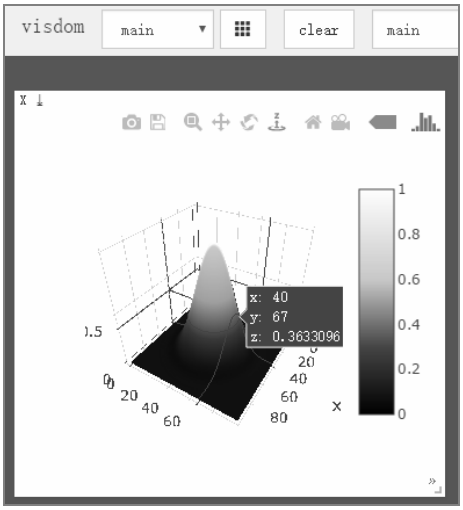

图 9-16　表面图

12．vis.contour

vis.contour 函数用来画轮廓图，其输入参数如下。

- X（tensor）：形状 $N \times M$，指定轮廓图中的值。

该函数支持以下特定选项。

- options.colormap：颜色谱，字符串格式，默认为“Viridis（翠绿色）”。
- options.xmin：设置最小值，数字格式，默认是 X 的最小值。
- options.xmax：设置最大值，数字格式，默认是 X 的最大值。

本案例文件名为 Chapter09/pt09_vm11_contour.py，代码如下。

```
from visdom import Visdom
import numpy as np
```

```
vis = Visdom()

x = np.tile(np.arange(1, 101), (100, 1))
y = x.transpose()
X = np.exp((((x - 50) ** 2) + ((y - 50) ** 2)) / -(20.0 ** 2))

vis.contour(X=X, opts=dict(colormap='Viridis'))
```

运行脚本，显示效果如图 9-17 所示。

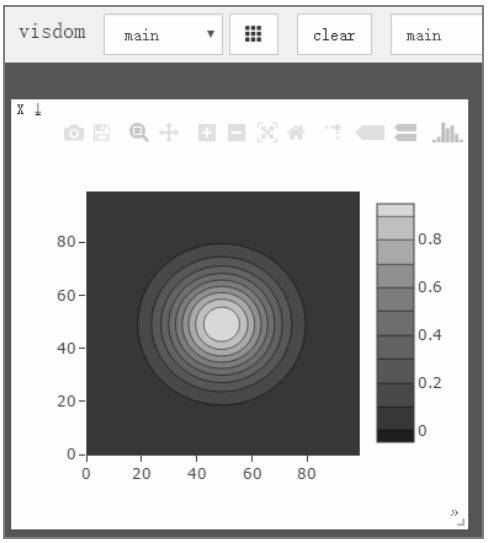

图 9-17　轮廓图

13．vis.mesh

vis.mesh 函数用来画网格图，其输入参数如下。

- X（tensor）：形状 *N*×2 或 *N*×3，定义 *N* 个顶点。
- Y（tensor，optional）：形状 *M*×2 或 *M*×3，定义多边形。

该函数支持下列特定选项。

- options.color：颜色（string）。
- options.opacity：多边形的不透明性（取值为 0～1）。

本案例文件名为 Chapter09/pt09_vm12_mesh.py，代码如下。

```
from visdom import Visdom
import numpy as np

vis = Visdom()

# mesh plot
x = [0, 0, 1, 1, 0, 0, 1, 1]
y = [0, 1, 1, 0, 0, 1, 1, 0]
z = [0, 0, 0, 0, 1, 1, 1, 1]
X = np.c_[x, y, z]
i = [7, 0, 0, 0, 4, 4, 6, 6, 4, 0, 3, 2]
j = [3, 4, 1, 2, 5, 6, 5, 2, 0, 1, 6, 3]
k = [0, 7, 2, 3, 6, 7, 1, 1, 5, 5, 7, 6]
Y = np.c_[i, j, k]

vis.mesh(X=X, Y=Y, opts=dict(opacity=0.5))
```

运行脚本，显示效果如图 9-18 所示。

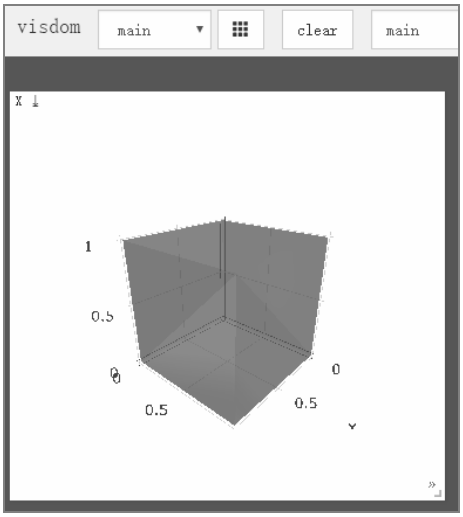

图 9-18　网格图

14．vis.svg

vis.svg 函数用来绘制 SVG 对象。该函数的输入是一个 SVG 字符串或一个 SVG

文件的名称。

本案例文件名为 Chapter09/pt09_vm13_svg.py，代码如下。

```
from visdom import Visdom
import numpy as np

vis = Visdom()

svgstr = """
<svg height="300" width="300">
  <ellipse cx="80" cy="80" rx="50" ry="30"
   style="fill:red;stroke:purple;stroke-width:2" />
  抱歉，你的浏览器不支持在线显示 SVG 对象.
</svg>
"""
vis.svg(
    svgstr=svgstr,
    opts=dict(title='SVG 图像')
)
```

运行脚本，显示效果如图 9-19 所示。

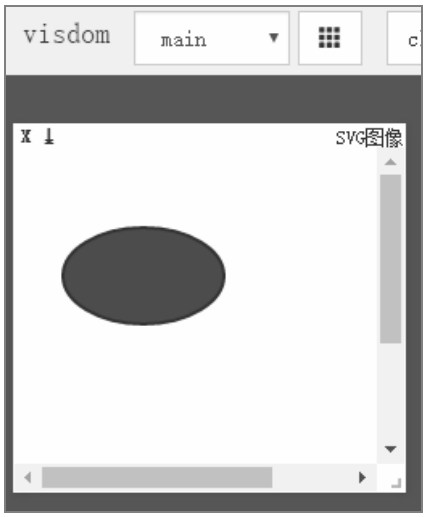

图 9-19　SVG 图像

9.2.6　Visdom 与 PyTorch

虽然 Visdom 强大的绘图功能可以与 PyTorch 结合在一起使用，但是截至 2018 年 2 月 10 日，PyTorch 仍然不支持 Windows 环境，仅支持 Linux 和 Mac OS 系统，而对于绝大部分入门级别的程序员来说，Linux 环境的门槛还是比较高的，所以本书没有提供 Visdom 与 PyTorch 结合的相关案例。Visdom 其实是一个和 Plotly 一样的可视化工具，其在机器学习中的应用也仅仅是用来可视化而已。读者如果想要自行研究 Visdom 与 PyTorch 结合的案例，可以参考 9.1 节 Plotly 在 Sklearn 中的应用的相关案例，对 PyTorch 的训练结果进行相应修改即可，前提是需要自行准备 Linux 环境。

第 10 章

Plotly 在量化投资中的应用

量化投资，简单地说，就是通过计算机编程的方法从历史数据中找到可以盈利的规律，并把它应用到未来数据上，在未来数据上实现盈利。与传统投资相比，量化投资最大的特点是在投资策略中广泛地应用程序化思想。

虽然量化投资的核心盈利策略与 Plotly 没什么关系，但是并不意味着 Plotly 就不能应用于量化投资中。实际上，任何投资策略的最终结果都需要一个漂亮的界面来呈现，这就是 Plotly 在量化投资中的意义。根据笔者的经验，投资策略结果的界面呈现只适合作为回测平台的一个扩展，如果读者没有开源的或自己写的回测平台，那么就没有必要为每个投资策略结果都单独设计一个输出界面，那样做无异于浪费时间。

本章的目的是给现有的回测平台适配基于 Plotly 的界面输出结果。本书使用的是 Python 3 的本地开发环境，这就限制了市场上的在线回测平台及基于 Python 2 的本地回测平台的使用。本章选择给回测平台 zwQuant 添加输出界面扩展，是因为 zwQuant 免费且开源，支持 Python 3，而且简单、易于上手，整个回测平台仅由几个.py 文件构成。

目前 zwQuant 只发布了第 1 版，其中有很多细节不够完善，本书提供的代码增加了对 Plotly 输出界面的支持，是一个类似网页的界面，并调整了 zwQuant 的一些细节，使之更容易扩展和使用。

在学习本章之前，有一点需要说明，本章内容只涉及对 zwQuant 的 Plotly 界面输出的封装，不涉及 zwQuant 的具体使用，读者如果对 zwQuant 感兴趣，可以参考《零起点 Python 大数据与量化交易》（以下简称《零起点》）一书，本章涉及的扩展方法对这本书的所有案例都兼容。使用本书提供的修改版 zwQuant 回测框架，对《零

起点》这本书涉及的策略文件稍微修改一下并运行即可得到 Plotly 的精美绘图结果。

运行 Chapter10/zq902_macd_v2.py 文件，结果如图 10-1 所示。

图 10-1　绘图结果

注意

Pandas 升级到 0.21.0 版本之后，其版本的兼容性与之前版本的兼容性有点差别，所以导致这个案例运行出错。这个错误的原因是 Pandas 团队在新的版本（0.21.0）中对 idxmax 函数的设计没有考虑周全。

所以，建议使用的 Pandas 版本不超过 0.20.xx。

这是对 zwQuant 适配 Plotly 扩展的界面，下面对这个界面进行解读。

在默认情况下，是不显示 Plotly 界面的，要开启 Plotly 界面的显示，首先需要对策略参数进行如下设置：

```
# 设置策略参数
qx.staVars=[12,26,'2015-01-01','']
```

```
    qx.debugMod=1
    qx.plotly_mode_flag = True
    qx.staFun=zwsta.macd20 # 绑定策略函数并运行回溯主函数
```

其中，qx.plotly_mode_flag=True 就是指开启 Plotly 输出模式。如果设置了 qx.plotly_mode_flag=True，在 bt_endRets 函数中会执行下面的代码：

```
    # 自定义输出结果
    if qx.plotly_mode_flag == True:
           zwdr.my_plotly_show(qx)
    else:
           zwdr.my_qunt_plot(qx)
```

zwdr.my_plotly_show()对应的 Chapter10/zwQTDraw.py 文件的 my_plotly_show()函数代码如下。

```
    import plotly as py
    import plotly.figure_factory as FF
    import plotly.graph_objs as go

    # ----------pre def
    pyplt = py.offline.plot

    def my_plotly_show(qx):
        list_result = qx.result_info
        list_result.append(['', ''])  # 为了能够凑够 24*2（原来 23*2），
        list0, list1, list2, list3 = [list_result[6 * i:6 * i + 6] for i in
range(0, 4)]
        arr_result = np.concatenate([list0, list1, list2, list3], axis=1)
        df_table = pd.DataFrame(arr_result, columns=['回测内容', '回测结果'] * 4)

        figure = FF.create_table(df_table, index_title='Date')
        _df = qx.qxLib
        trace1 = go.Scatter(
            x=_df['date'],
            y=_df['val'],
            name='val',
            xaxis='x2',
            yaxis='y2',
            legendgroup='one',
            # mode = 'lines',
            line=dict(
```

```
            color=('#FF0000'),
            width=3),
    )

    trace2 = go.Scatter(
        x=_df['date'],
        y=_df['dret'],
        xaxis='x2',
        yaxis='y3',
        name='dret',
        legendgroup='one',
        line=dict(
            color='#00EE00',
            width=1)
    )

    # 把 trace 添加到 figure 中
    figure['data'].extend(go.Data([trace1, trace2]))

    figure.layout.update(
        yaxis={'domain': [.7, 1]},
        yaxis2={'domain': [0, .65],
                'anchor': 'x2',
                'title': '剩余资金'},
        yaxis3=dict(
            title='收益率',
            titlefont=dict(
                color='rgb(32, 253, 1)'
            ),
            tickfont=dict(
                color='rgb(32, 253, 1)'
            ),
            overlaying='y2',
            side='right'
        ),
        xaxis2={'anchor': 'y2'},
        margin={'t': 50, 'b': 30, 'l': 50, 'r': 50},
        title='回溯测试输出结果',
        height=1000,
        legend=dict(x=.9,
                    y=.6,
```

```
                    xanchor='auto',
                    yanchor='auto')
    )

    # 画图!
    pyplt(figure, filename='tmp/{}.html'.format(qx.prjName
), show_link=False)
```

运行结果如图 10-2 所示。

图 10-2　绘图结果

由于纸质版的原因，这幅图的动态效果在这里体现不出来，如果想要看它的原始结果，可以使用浏览器打开本地文件 Chapter10\tmp\macd20.html，在计算机中查看动态绘图结果。在认识 Plotly 之前，笔者印象中要想做出这种效果，只有老老实实地去学习网页开发的一些框架，例如 Django、Flask 等。虽然技多不压身，多学点网页开发技术对自己有好处，但是也要考虑实际需求与时间成本。从笔者金融行业背景的角度来说，仅是想要一个简单的输出界面，没有必要也没有动力去学习那

么多网页开发的基础，因此，Plotly 的出现完全满足了个人的需求。下面就开始对图 10-2 对应的代码进行分析。

从绘图结果中可以发现，这个绘图结果是一个表格与一个具有主次坐标轴的直线图的混合结果。通过前面所学到的知识，我们知道单独绘制一个表格很简单，而绘制主次坐标轴的直线图也不是太麻烦的事情，但是要把两者结合在一起就有一些难度了，接下来就解决这个问题。

首先，新建一个表格。传入的 qx.result_info 是一个 23×2 结构的列表，我们添加一个 1×2 的元素，把它转换成 24×2 结构的列表；然后把这个列表转换成 6×8 结构的 NumPy 数组；最后把这个数组转换成 6×8 结构的 DataFrame，利用 create_table 函数把 DataFrame 转换成 Plotly 的表格，代码如下。

```
    list_result = qx.result_info
    list_result.append(['', ''])  # 为了能够凑够 24*2（原来 23*2），
    list0, list1, list2, list3 = [list_result[6 * i:6 * i + 6] for i in range(0,
4)]
    arr_result = np.concatenate([list0, list1, list2, list3], axis=1)
    df_table = pd.DataFrame(arr_result, columns=['回测内容', '回测结果'] * 4)

    figure = FF.create_table(df_table, index_title='Date')
```

接下来，新建两个 trace，用来构建拥有主次坐标轴的线条图，代码如下。

```
    _df = qx.qxLib
    trace1 = go.Scatter(
        x=_df['date'],
        y=_df['val'],
        name='val',
        xaxis='x2',
        yaxis='y2',
        legendgroup='one',
        # mode = 'lines',
        line=dict(
            color=('#FF0000'),
            width=3),
    )

    trace2 = go.Scatter(
        x=_df['date'],
```

```
        y=_df['dret'],
        xaxis='x2',
        yaxis='y3',
        name='dret',
        legendgroup='one',
        line=dict(
            color='#00EE00',
            width=1)
    )
```

qx.qxLib 存放的是每日交易情况的汇总，如果读者很想知道它到底长什么样子，可以用 Excel 打开文件 Chapter10\tmp\macd20_qxLib.csv。每次回测之后，qx.qxLib 都会在本地保存一份，这个文件就是本次回测的备份结果。

上面的代码并不难理解，只是有几个参数需要在这里解释一下。

对于 trace1 来说，如下几个参数比较重要。

- xaxis='x2'，表示 trace1 是在第二幅图上绘制的（第一幅图对应参数值默认是 x1）。
- yaxis='y2'，解释同上。
- legendgroup='one'，默认是''，如果赋值为 string（这里是'one'），那么对于所有 legendgroup='one'的 legend 来说，单击任意一个 legend，等价于单击该 legendgroup 的所有 legend。

对于 trace2 来说，与 trace1 类似，只有下面这个参数稍微不同。

- yaxis='y3'，这里之所以要设置为'y3'，而不是'y2'，是因为'y2'要被设置为第二幅图的主坐标轴，因此，次坐标轴就属于'y3'了。

接下来，把 trace1、trace2 合并到表格里面，代码如下。

```
    # 把 trace 添加到 figure 中
    figure['data'].extend(go.Data([trace1, trace2]))
```

最后，由于 figure['data']里面既有之前创建的 table，又有 trace1 和 trace2，如果不对它们进行布局，Plotly 会默认把它们混合在一起，所以需要对 figure 进行布局，代码如下。

```
    figure.layout.update(
        yaxis={'domain': [.7, 1]},
        yaxis2={'domain': [0, .65],
                'anchor': 'x2',
                'title': '剩余资金'},
```

```
        yaxis3=dict(
            title='收益率',
            titlefont=dict(
                color='rgb(32, 253, 1)'
            ),
            tickfont=dict(
                color='rgb(32, 253, 1)'
            ),
            overlaying='y2',
            side='right'
        ),
        xaxis2={'anchor': 'y2'},
        margin={'t': 50, 'b': 30, 'l': 50, 'r': 50},
        title='回溯测试输出结果',
        height=1000,
        legend=dict(x=.9,
                    y=.6,
                    xanchor='auto',
                    yanchor='auto')
    )
```

上面的代码有些凌乱，下面分开解读。

对于 *yaxis* 轴，由于其是一个表格，我们希望它占据一定的空间高度，代码如下。

```
    yaxis={'domain': [.7, 1]},
```

对于 *yaxis2* 轴，由于其是线形图的主坐标轴，我们也希望它占据一定的空间高度，并且希望它对应 axis2，因为默认情况下对应的是 axis，代码如下。

```
    yaxis2={'domain': [0, .65],
                'anchor': 'x2',
                'title': '剩余资金'},
```

对于 *yaxis3* 轴，我们希望它和 *yaxis2* 轴一样都是坐标轴（overlaying='y2'），只是在右侧作为副坐标轴（side='right'）。另外，这里还用到了 tickfont 与 titlefont 参数，用来设置标题与刻度的颜色，代码如下。

```
    yaxis3=dict(
            title='收益率',
            titlefont=dict(
                color='rgb(32, 253, 1)'
```

```
            ),
            tickfont=dict(
                color='rgb(32, 253, 1)'
            ),
            overlaying='y2',
            side='right'
        ),
```

对于 *xaxis2* 轴，我们希望它能对应 *yaxis2* 轴，代码如下。

```
    xaxis2={'anchor': 'y2'},
```

然后设置窗口的举例（margin 参数）、figure 的标题（title 参数）、高度（height 参数）、图例（legend 参数）。注意，legend 参数中的意思是把图例的位置放在（0.9,0.6）的相对空间位置上，xanchor 与 yanchor 分别设置为'auto'表示程序会自动在水平与垂直方向上调整大小，结果会影响 margin 参数的效果，读者可以注释掉这两行代码，然后运行一下，查看前后的结果，代码如下。

```
    margin={'t': 50, 'b': 30, 'l': 50, 'r': 50},
           title='回溯测试输出结果',
           height=1000,
           legend=dict(x=.9,
                       y=.6,
                       xanchor='auto',
                       yanchor='auto')
```

至此，本章内容最重要的部分就介绍完毕了，如果读者对本章前面的内容已经了解了，会发现本章的内容是很简单的。

Plotly 的出现为金融数据可视化尤其是量化投资的结果可视化打开了一个新的天地：利用 Plotly，我们可以以很简单的方式绘制一些精美的动态图表；可以绘制类似网页的动态输出结果，而不需要学习一些网络开发框架，如 Django 和 Flask 等；可以把绘图结果原汁原味地分享给自己的朋友或同事，和他们一起分享我们的喜悦。总之，Plotly 的出现确实可以影响我们对数据可视化的固有观念，让我们的编程生活更加美好。

第 11 章

Plotly 在其他语言中的应用

11.1 Plotly在R语言中的应用

R 语言是用于统计分析、图形表示和报告的编程语言和软件环境。R 语言由 Ross Ihaka 和 Robert Gentleman 等在新西兰奥克兰大学创建，目前由 R 语言开发核心团队开发。R 语言问世以来，其官方社区为 R 语言提供了非常多的免费、开源而且强大的模块，这些模块的开发者很多都是学术界的“大牛”，并且紧跟学术前沿，因此 R 语言在学术界有着非常高的地位。

R 语言由于其安装空间小、模块管理方便、易学易用等特点，在数据分析与处理方面有着很重要的地位。使用 R 语言绘图也非常方便，如大名鼎鼎的 ggplot 可以说是 R 语言的绘图神器。但是和 python 一样，这些绘图模块绘制的图表都是静态的，而 Plotly 是基于 JS 的动态绘图模块，本节主要介绍 Plotly 在 R 语言中的应用。

11.1.1 安装 R 语言

R 语言支持 Mac、Linux 和 Windows 三大系统，我们可以从 R 语言的官方网站（https://cran.r-project.org/）下载对应系统的安装包，这里以 Windows 操作系统为例

进行介绍。

从 R 语言官方网站下载 R 的 Windows 安装包，并将其保存在本地目录中。它是一个名为“R-version-win.exe”的 Windows 安装程序（.exe），只需双击并运行安装程序，接受默认设置即可。如果读者的 Windows 是 32 位版本，那么它将安装 32 位版本；如果读者的 Windows 是 64 位版本，那么它将安装 32 位和 64 位版本。

安装后，既可以单击桌面上的程序快捷图标运行程序，也可以单击 Windows 程序文件下的路径“R\R3.2.2\bin\i386\Rgui.exe”运行程序。这时会打开 R-GUI，这是一个 R 控制台，可以在里面进行 R 语言编程。

11.1.2 安装 Plotly 模块

安装 Plotly 模块最简单的方法是在控制台中通过以下命令安装：

```
install.packages("plotly")
```

安装完成后，在使用这个包里面的函数之前要加载 Plotly 包，代码如下。

```
library(plotly)
```

在这个包里面有一个非常重要的函数 plot_ly，它是用来绘制图形的函数。由于 R 语言的绘图并不是本书的重点，所以本书仅介绍几个入门图形的绘制方法。

11.1.3 Plotly 应用分析

本节主要介绍 Plotly 在 R 语言中的基本应用，本节的所有代码都可以在文件 code.R 中找到。

1. 散点图

使用 plot_ly 函数画几幅简单的图形，最常见的就是散点图。本案例使用模拟的数据进行画图，代码如下。

```
library(plotly)
set.seed(123)
x1<-rnorm(50, mean = 0, sd = 1)
```

```
x2<-rnorm(50, mean = 1, sd = 1)
data<-data.frame(x1, x2)
plot_ly(data, x = ~x1, y = ~x2, type ="scatter")
```

绘图结果如图 11-1 所示。

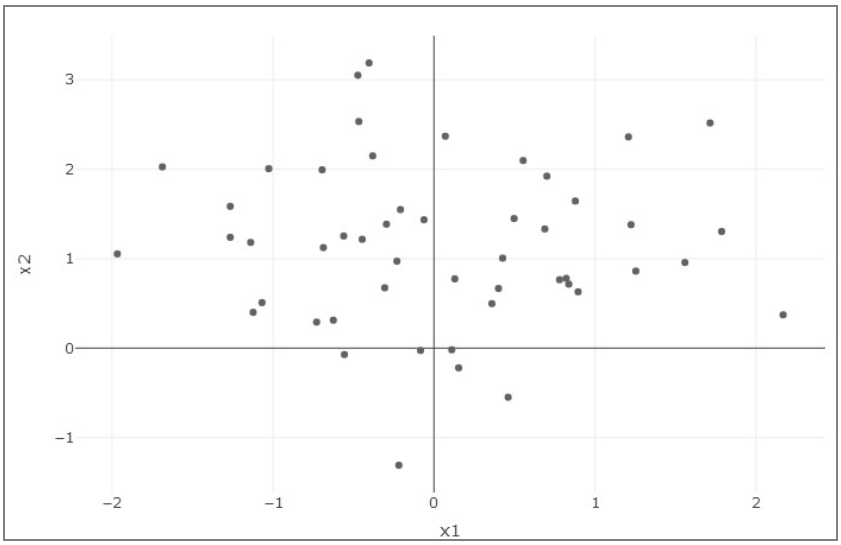

图 11-1　散点图

代码解释如下。

- 第 1 行代码表示加载 Plotly 包，加载后可以一直使用 Plotly 包里面的函数，如果 R 语言关闭后重新打开，则需要再加载一次该包。
- 第 2 行代码表示设定随机种子，设定随机种子后，每次产生的随机数将是一样的。
- 第 3 行代码表示产生样本量为 50、均值为 0、标准差为 1 的标准正态分布随机数 x1。
- 第 4 行代码表示产生样本量为 50、均值为 1、标准差为 1 的标准正态分布随机数 x2。
- 第 5 行代码表示将产生的两个随机数放入数据框 data 中（因为 plot_ly 函数要求 data 是数据框格式）。
- 第 6 行代码表示画 x1 与 x2 的散点图，其中，x = ～x1 表示以 x1 为 *X* 轴（横坐标），y = ～x2 表示以 x2 为 *Y* 轴（纵坐标），type ="scatter"表示画图的类型为散点图。

注意

（1）如果读者用的是 R 语言，以上代码画的图形会在读者的浏览器网页上显示，如果读者用的是 RStudio，则图形会在 RStudio 自带的浏览器中显示。

（2）与 Python 不同，Plotly 在 R 语言中的绘图没有离线绘图与在线绘图之说，所有绘图都是离线绘图。因此，本节内容就没有区分离线绘图与在线绘图。

2. 散点图（三维）

使用 plot_ly 函数绘制三维散点图也是很方便的，代码如下。

```
set.seed(123)
x1<-rnorm(20, mean = 0, sd = 1)
x2<-rnorm(20, mean = 1, sd = 1)
x3<-rnorm(20, mean = 2, sd = 1)
data<-data.frame(x1, x2,x3)
plot_ly(data, x = ~x1, y = ~x2,z=~x3,type ="scatter3d")
```

这里的代码与二维散点图中的代码的区别在于 type 这个参数，只需要把 scatter 换成 scatter3d 即可。绘制的结果如图 11-2 所示。

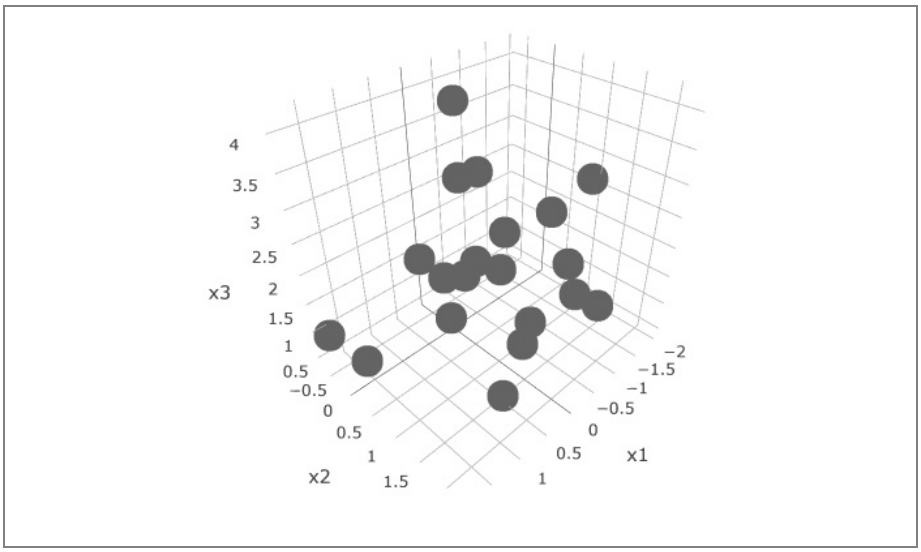

图 11-2　三维散点图

3．线形图

接下来介绍一些基本线形图的画法。绘制线形图同样使用 plot_ly 函数，type 参数中没有提供线形的类型，因为在 Plotly 里面，线形图是由散点图演变过来的，我们可以在函数里面添加参数 mode 把散点图的点连接成线，代码如下。

```
set.seed(123)
x <- seq(1,50,1)
y <- rnorm(50, mean = 0, sd = 1)
data <- data.frame(x, y)
plot_ly(data, x = ~x, y =~ y, type = "scatter", mode = "lines")
```

代码解释如下。

- 第 2 行 seq 函数表示生成有规律的序列，我们在此生成从 1 到 50、间隔为 1 的 50 个数据。
- 第 3 行代码表示产生样本量为 50、均值为 0、标准差为 1 的标准正态分布随机数 y。
- 第 4 行代码表示将数据 x、y 放入数据框 data 中。
- 第 5 行代码表示以 x 为横轴、以 y 为纵轴画线形图。

绘制结果如图 11-3 所示。

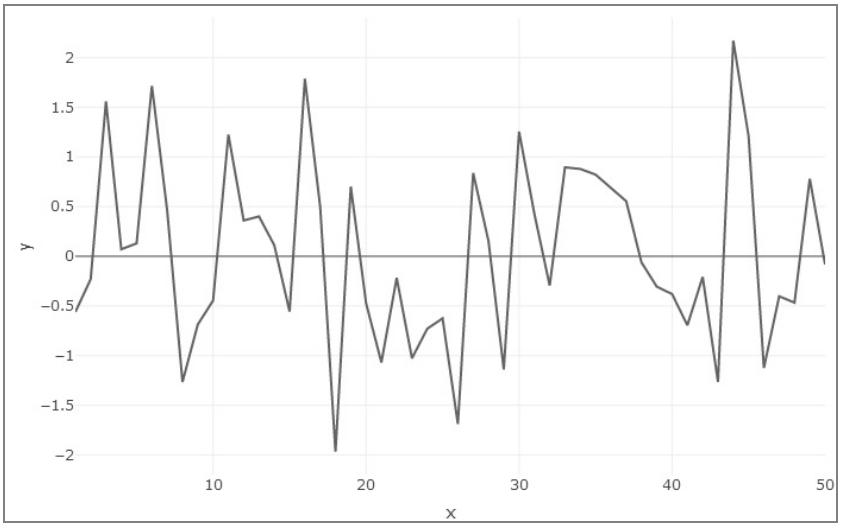

图 11-3　线形图

在图 11-3 中我们只画了一条线，也可以在同一幅图中画多条线，有两种方法可以完成这个工作。接下来用两种方法在同一幅图中画出三条线。

第一种方法：先画出一条线，然后在该图形中添加另外两条线。

第二种方法：先画出坐标轴（画出横轴），然后在坐标轴中添加三条线。

4．线形图（多元1）

```
    set.seed(123)
    y1 <- rnorm(50, mean = 0, sd = 1)
    y2 <- rnorm(50, mean = 3, sd = 1)
    y3 <- rnorm(50, mean = 6, sd = 1)
    x <- seq(1,50,1)
    data <- data.frame(x, y1, y2, y3)
    plot_ly(data, x = ~x, y = ~y1, name = "y1", type ="scatter", mode ="lines")
%>%
    add_trace(y = ~y2, name = "y2", mode = "lines+markers") %>%
    add_trace(y = ~y3, name = "y3",mode = "markers")
```

代码解释如下。

- 第 1 行代码表示设定随机种子。
- 第 2、3、4 行代码表示生成随机数，我们在此设定均值分别为 0、3、6，主要是为了便于观察图形。
- 第 5 行代码表示产生 1 到 50、间隔为 1 的序列。
- 第 6 行代码表示将 x、y1、y2、y3 放到数据框 data 中。
- 第 7 行代码表示以 x 为横坐标轴、以 y1 为纵坐标轴画线形图。
- 第 8 行代码表示在原坐标轴上添加曲线 y2。
- 第 9 行代码表示在原坐标轴上添加曲线 y3。

注意

第 7、8、9 行中的 name 参数表示将曲线 y1 命名为 y1、将曲线 y2 命名为 y2、将曲线 y3 命名为 y3。

参数 mode 表示线条类型，lines 表示加线，markers 表示为对应的点加上标记，它们可以同时使用，即 lines+markers。

%>%可以表示一个连接符，连接符后面还有语句，直到语句后面不再出现

%>%时，执行从开始出现%>%到结束的语句。

绘图结果如图 11-4 所示。

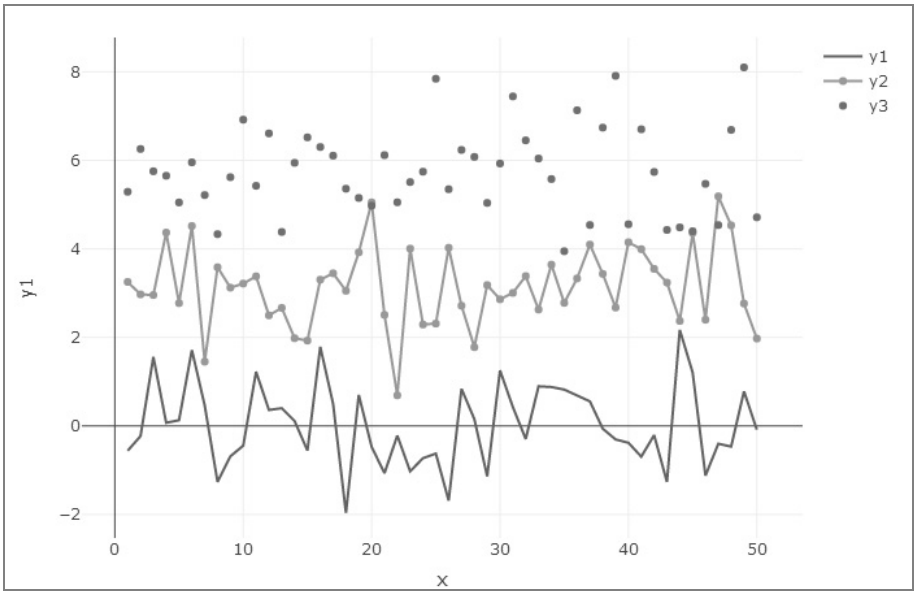

图 11-4　三条曲线图

5. 线形图（多元2）

```
set.seed(123)
y1 <- rnorm(50, mean = 0, sd = 1)
y2 <- rnorm(50, mean = 3, sd = 1)
y3 <- rnorm(50, mean = 6, sd = 1)
x <- seq(1,50,1)
data <- data.frame(x, y1, y2, y3)
plot_ly(data, x = ~x) %>%
add_trace(y = ~y1, name = "y1",mode = "lines") %>%
add_trace(y = ~y2, name = "y2", mode ="lines+markers") %>%
add_trace(y = ~y3, name = "y3", mode = "markers")
```

绘图结果如图 11-5 所示。

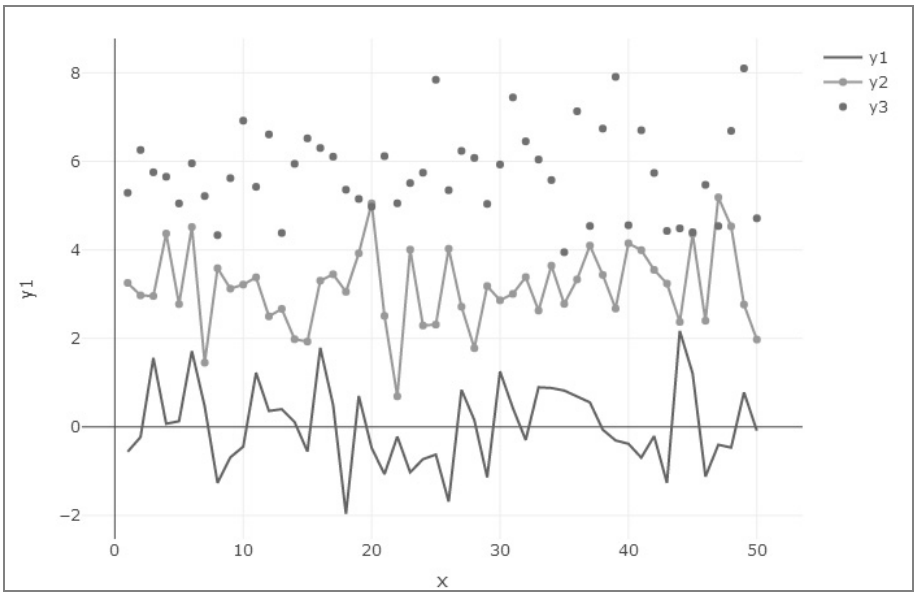

图 11-5　三条曲线图

从图 11-4 和图 11-5 可以看出，上面两种方法绘图的结果是一样的，只是实现的方法不一样。

11.1.4　更多扩展

使用 plot_ly 函数可以画多种类型的图，比较简单的方法是通过修改参数 type 来实现，type 的可选参数有："scatter"、"box"、"bar"、"heatmap"、"histogram"、"histogram2d"、"histogram2dcontour"、"pie"、"contour"、"scatterternary"、"sankey"、"scatter3d"、"surface"、"mesh3d"、"scattergeo"、"choropleth"、"scattergl"、"pointcloud"、"heatmapgl"、"parcoords"、"scattermapbox"、"carpet"、"scattercarpet"、"contourcarpet"、"ohlc"、"candlestick"和"area"。具体的使用方法及案例可参考 Plotly 中的 plot_ly 函数的使用方法。

如果读者对 Plotly 在 R 语言中的应用有更多的需求，可以参考官方网站 https://plot.ly/r/，在这个网站中可以找到绝大部分想要的案例，官网部分截图如图 11-6 所示。

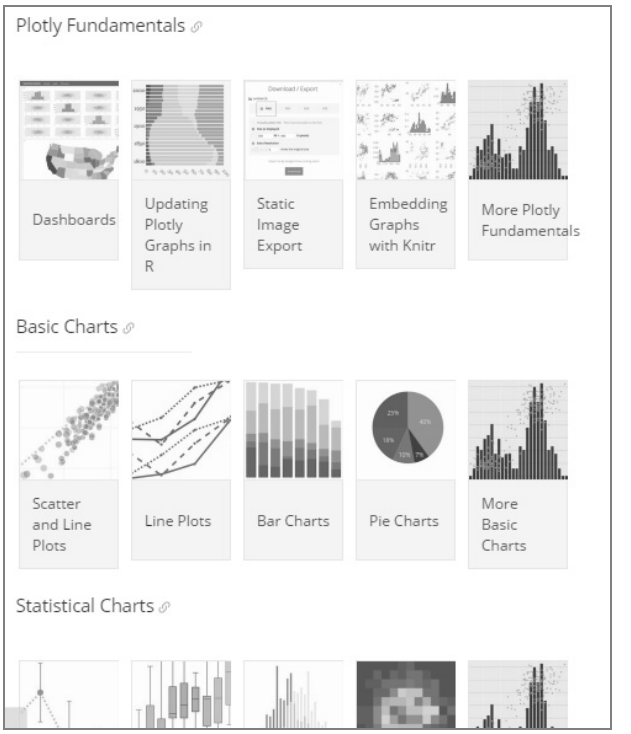

图 11-6　官网部分截图

11.2　Plotly在MATLAB中的应用

Plotly 作为强大的数据可视化工具，拥有支持 MATLAB 使用的接口，对于众多使用 MATLAB 进行研究工作的用户来说，能将 MATLAB 强大的数据处理能力与 Plotly 的可视化功能结合起来，无疑是一个巨大的帮助。由于 Plotly 绘图模块丰富的内容和优秀的美化功能，使得 Plotly 在 MATLAB 中得到了广泛应用。

11.2.1　下载与安装

MATLAB 的 Plotly 工具箱可以从官网目录 https://plot.ly/matlab/下的 Getting Started 中下载，下载后将其解压，得到一个名为 MATLAB-Online-master 的文件夹，

然后打开 MATLAB 进行 Plotly 工具箱的安装。

```
>> cd ~\Download\MATLAB-Online-master
>> %将解压的文件夹设置为 current folder
>> plotlysetup('Username', 'API Key')
```

其中，Username 和 API Key 可以在官网上免费注册获得，笔者为本书注册的 Username 为 PlotlyBookTest，API Key 为 ECmqAy8kLE5Qk7h29trH。

安装成功后将显示 Plotly 工具箱已成功安装并导入 MATLAB 路径中。

```
Adding Plotly to MATLAB toolbox directory ...  Done
Saving Plotly to MATLAB search path via startup.m ...  Done
Saving username/api_key credentials ... Done

Welcome to Plotly! If you are new to Plotly please enter: >> plotlyhelp
to get started!
```

安装完成后可以将上述下载并解压得到的文件夹删除。

接下来便可以使用 Plotly 工具箱在线画图了！

注意

（1）如果下一次重新打开 MATLAB 时遇到如图 11-7 所示的警告，可打开 D:\software\Matlab2013b\1\toolbox\local\startup.m 文件，将里面的“addpath(genpath(fullfile(matlabroot,'toolbox','plotly')),'-end');”语句剪切至 function…end 之间保存，然后重新启动 MATLAB。

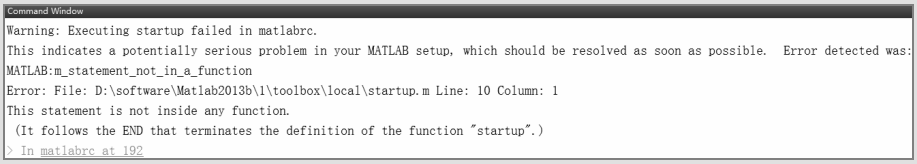
Command Window
Warning: Executing startup failed in matlabrc.
This indicates a potentially serious problem in your MATLAB setup, which should be resolved as soon as possible. Error detected was:
MATLAB:m_statement_not_in_a_function
Error: File: D:\software\Matlab2013b\1\toolbox\local\startup.m Line: 10 Column: 1
This statement is not inside any function.
(It follows the END that terminates the definition of the function "startup".)
> In matlabrc at 192

图 11-7　警告

（2）需要版本更新时运行 plotlyupdate 文件，但此功能可能会间接性失效，失效时只能通过官网下载最新版本，再按照上述流程安装一遍。

11.2.2　基础入门

1．在线绘图

接下来利用 Plotly 将 MATLAB 中的图变成既美观又实用的交互式在线图片，只需要调用一个函数 fig2plotly()即可，代码如下。

```
[X, Y, Z] = peaks;
contour(X, Y, Z, 20);
Plotly_figure = fig2plotly()
```

运行后将弹出一个网页，显示调用 Plotly 后的绘图结果，并将图片保存在账户的云端。此运行结果如图 11-8 所示。

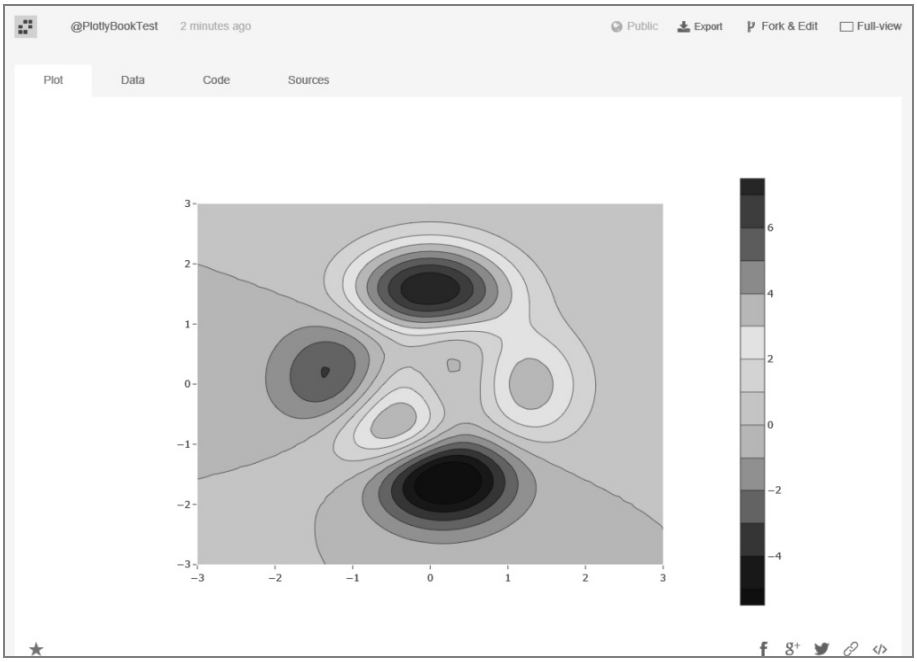

图 11-8　在线绘图结果

若不希望 Plotly 对图片进行调色改变，希望保留 MATLAB 中图片的样式，则可以使用 fig2plotly（gcf,'strip',false）。另外，可通过所得到结构体 plotly_fig 中的 layout 和 PlotOptions 中的参数进行其他设置，如 fig2plotly（gcf,'filename', 'plotly1'）设置

文件名等。

2．离线绘图

在线绘图的速度过慢，我们更常用的是 Plotly 支持的离线绘图功能。

要使用 Plotly 的离线绘图功能，首先需要下载一个离线绘图包。此下载流程已在 Plotly 工具箱中包含了，只需要运行下面这行代码即可。

```
>>getplotlyoffline('https://cdn.plot.ly/plotly-latest.min.js')
```

下载成功后输出以下代码。

```
Success! You can generate your first offline graph
using the 'offline' flag of fig2plotly as follows:

>> plot(1:10); fig2plotly(gcf, 'offline', true);
```

注意

若下载时报错，可以先尝试运行后面的离线绘图代码，也许已经可以正常使用了。

接下来就可以使用 Plotly 的离线绘图功能了，使用离线绘图只需要在在线绘图代码上稍作修改，将'offline'参数设置为 true，代码如下。

```
[X, Y, Z] = peaks;
contour(X, Y, Z, 20);
plotly_fig = fig2plotly(gcf, 'offline',true, 'filename',
'plotly_offline')
```

此时绘图结果将形成一个 HTML 文件保存在当前文件夹下。将 plotly_fig.url 复制并粘贴至浏览器的地址栏中，即可查看保存在本地的绘图结果，代码如下。

```
>> plotly_fig.url
ans =
file://C:\Users\Documents\Plotly\plotly_offline.html
```

离线绘图的运行结果如图 11-9 所示。

若离线绘图的操作已经全部完成，希望将结果上传至云端，只需要单击图 11-9 右上角的“保存并上传”按钮即可。

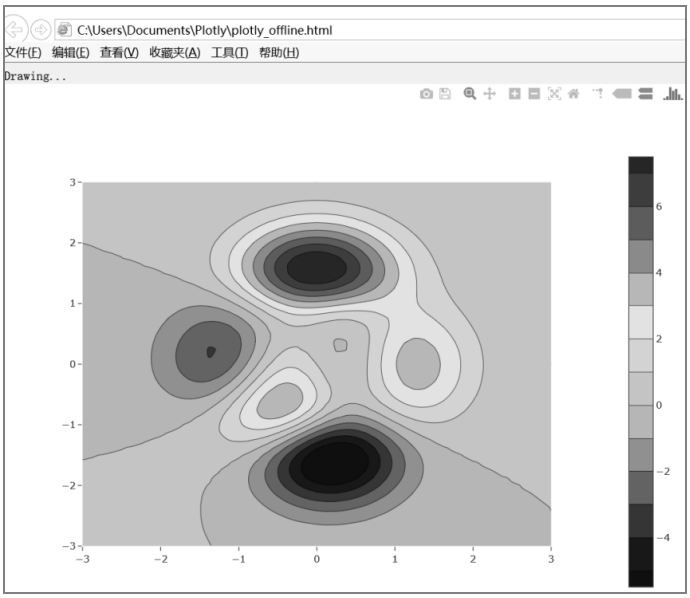

图 11-9　离线绘图结果

至此，已经学习完 MATLAB 操作 Plotly 工具箱的基本操作了，接下来是一些更复杂的经典案例，重点介绍离线绘图操作。

11.2.3　经典案例

1. 气泡散点图

（1）简单气泡散点图

首先设置坐标、大小、颜色并在 MATLAB 中绘图，然后使用 Plotly，代码如下。

```
x = 50 * randn(1, 30);
y = 50 * randn(size(x));
r = randi(1000, size(x));
c = randi(10, size(x));

fig = figure;
scatter(x, y, r, c, 'filled', 'MarkerEdgeColor', 'k')
% finish plotting on your computer first
```

```
    % --PLOTLY-- %
    % strip = false => preserve MATLAB style
    % offline = true => use plotly offline
    response = fig2plotly(fig, 'strip', false, 'offline', true, 'filename',
'MATLAB-bubble-chart');
    plotly_url = response.url;
```

绘图结果如图 11-10 所示。

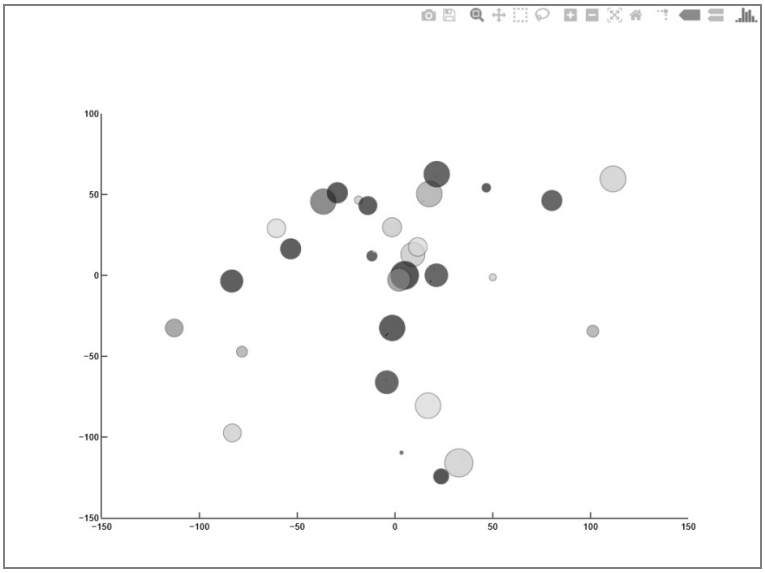

图 11-10　简单气泡散点图

（2）复合气泡散点图

可以同时对多份数据绘图，并画出多种不同形状的气泡，也可以不通过 MATLAB 而直接利用 Plotly 绘图，代码如下。

```
    trace1 = struct(...
        'x', [1, 2, 3, 4], 'y', [10, 11, 12, 13], ...
        'mode', 'markers', 'marker', struct('size', [12, 22, 32, 42],...
          'color',{{'hsl(0,100,40)','hsl(33,100,40)','hsl(66,100,40)',
'hsl(99,100,40)'}}, ...
          'opacity', [0.6, 0.7, 0.8, 0.9]), ...
        'type', 'scatter');
```

```
    trace2 = struct(...
        'x', [1, 2, 3, 4], 'y', [11, 12, 13, 14], ...
        'mode', 'markers', 'marker', struct( 'color', 'rgb(31, 119, 180)', ...
            'size', 18, 'symbol', { {'circle', 'square', 'diamond',
'cross'}}), ...
        'type', 'scatter');
    trace3 = struct(...
        'x', [1, 2, 3, 4], 'y', [12, 13, 14, 15], ...
        'mode', 'markers', 'marker', struct('size', 18, 'line', struct(...
          'color',{{'rgb(120,120,120)','rgb(120,120,120)','red',
'rgb(120,120,120)'}}, ...
          'width', [2, 2, 6, 2])), ...
        'type', 'scatter');

    % prepare the data
    data = {trace1, trace2, trace3};
    % layout : plot parameters
    layout = struct('showlegend',false, 'width', 800, 'height', 650); % width
and height are required parameters

    % plotting offline
    p = plotlyfig;
    p.data = data;
    p.layout = layout;
    p.PlotOptions.FileName = 'bubblechart';
    plotlyfig.PlotOptions.FileOpt = 'overwrite';
    response = plotlyoffline(p);

    %%% if you want to plot online, use the following code instead of "plotting
offline"
    % % response = plotly(data, struct('layout', layout, 'filename',
'bubblechart', 'fileopt', 'overwrite'));
    % % plotly_url = response.url
```

绘图结果如图 11-11 所示。

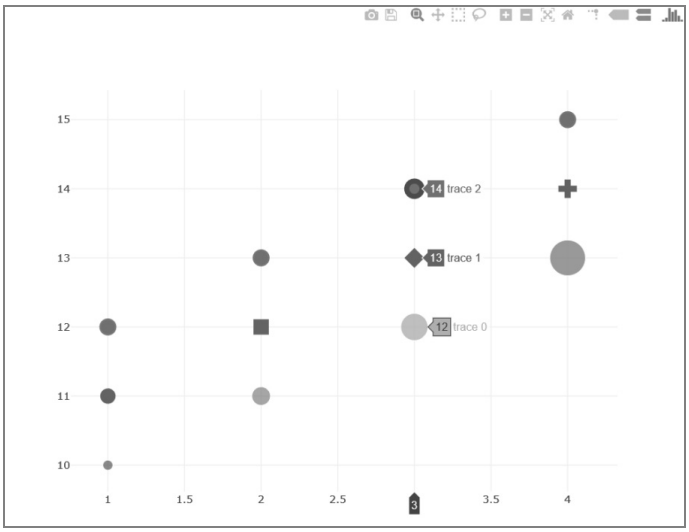

图 11-11　复合气泡散点图

2．时间序列

时间序列数据是金融领域重要的数据类型，Plotly 工具箱中有强大的功能来辅助数据可视化。

（1）标准日期

标准日期绘图案例的代码如下。

```
x = linspace(datenum(2009,01,01),datenum(2010,01,01),12);
y = rand(1,12);
plotlydate = convertDate(x);
plot(plotlydate,y);
plotlyfig = fig2plotly(gcf, 'offline', true, 'filename', 'timeseries');
plotlyfig.layout.xaxis1.type = 'date';
plotlyfig.PlotOptions.FileOpt = 'overwrite';
plotlyoffline(plotlyfig);
```

绘图结果如图 11-12 所示。

如果使用在线绘图，只需要将'offline'参数设为 false，将 plotlyoffline()函数替换为 plotly()函数。

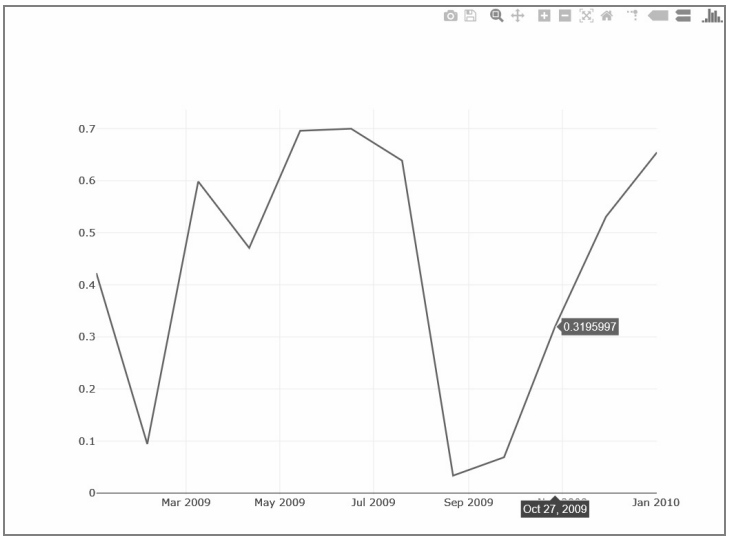

图 11-12　标准日期绘图

（2）字符串日期

字符串日期绘图案例的代码如下。

```
    data = {...
      struct(...
        'x', { {'2013-10-04 22:23:00', '2013-11-04 22:23:00', '2013-12-04
22:23:00'} }, ...
        'y', [1, 3, 6], ...
        'type', 'scatter')...
    };

    layout = struct('width', 800, 'height', 650); % width and height are
required parameters
    p = plotlyfig;
    p.data = data;
    p.layout = layout;
    p.PlotOptions.FileName = 'date-axes';
    plotlyfig.PlotOptions.FileOpt = 'overwrite';
    response = plotlyoffline(p)

    % 要使用 plotly()在线绘图，可参考复合气泡散点图代码的最后部分
```

绘图结果如图 11-13 所示。

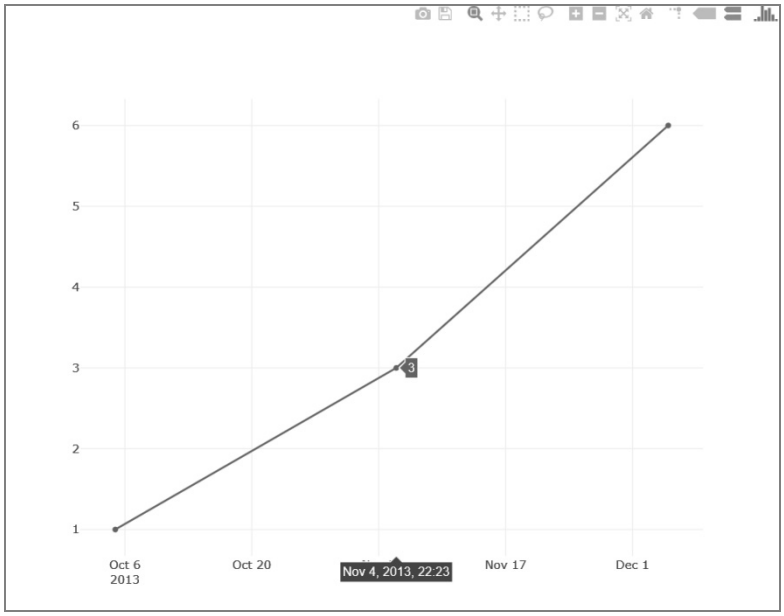

图 11-13　字符串日期绘图

3．频率直方图

下面介绍复杂一些的绘图，使用 Plotly 工具箱画两幅重叠的频率直方图，代码如下。

```
x0 = randn(500,1);
x1 = randn(500,1)+1;

trace1 = struct(...
  'x', x0, ...
  'opacity', 0.75, ...
  'type', 'histogram');
trace2 = struct(...
  'x', x1, ...
  'opacity', 0.75, ...
  'type', 'histogram');
data = {trace1, trace2};
layout = struct('barmode', 'overlay', 'width', 800, 'height', 650);
```

```
p = plotlyfig;
p.data = data;
p.layout = layout;
p.PlotOptions.FileName = 'overlaid-histogram';
plotlyfig.PlotOptions.FileOpt = 'overwrite';

response = plotlyoffline(p)
% 要使用 plotly()在线绘图，可参考复合气泡散点图代码的最后部分
```

绘图结果如图 11-14 所示。

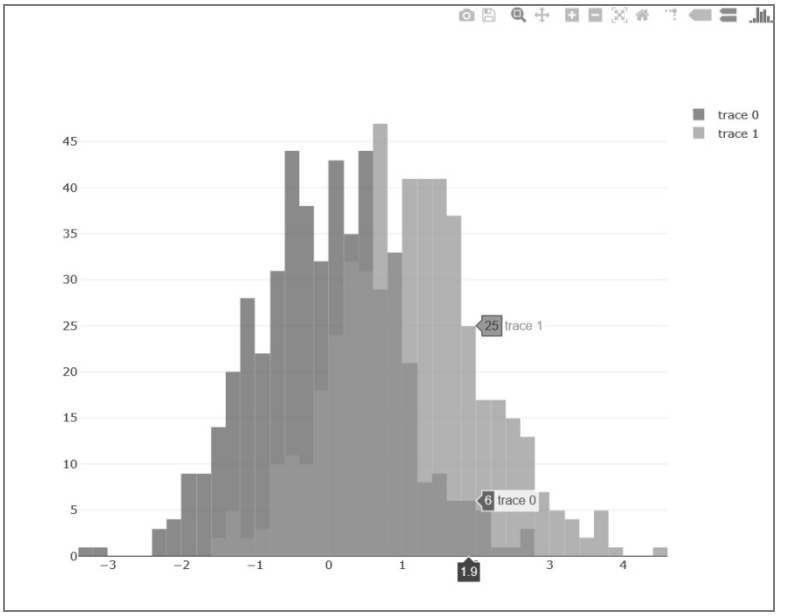

图 11-14　频率直方图

4．箱型图

箱型图在数据信息展示中的应用十分广泛，其包含的数据特征很多，绘图案例的代码如下。

```
x = [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2];
trace1 = struct(...
  'y', [0.2, 0.2, 0.6, 1.0, 0.5, 0.4, 0.2, 0.7, 0.9, 0.1, 0.5, 0.3], ...
```

```
  'x', x, ...
  'name', 'kale', ...
  'marker', struct('color', '#3D9970'), ...
  'type', 'box');
trace2 = struct(...
  'y', [0.6, 0.7, 0.3, 0.6, 0.0, 0.5, 0.7, 0.9, 0.5, 0.8, 0.7, 0.2], ...
  'x', x, ...
  'name', 'radishes', ...
  'marker', struct('color', '#FF4136'), ...
  'type', 'box');
trace3 = struct(...
  'y', [0.1, 0.3, 0.1, 0.9, 0.6, 0.6, 0.9, 1.0, 0.3, 0.6, 0.8, 0.5], ...
  'x', x, ...
  'name', 'carrots', ...
  'marker', struct('color', '#FF851B'), ...
  'type', 'box');
data = {trace1, trace2, trace3};
layout = struct(...
    'yaxis', struct(...
      'title', 'normalized moisture', ...
      'zeroline', false), ...
     'boxmode', 'group', 'width', 800, 'height', 650);

p = plotlyfig;
p.data = data;
p.layout = layout;
p.PlotOptions.FileName = 'box-grouped';
plotlyfig.PlotOptions.FileOpt = 'overwrite';
response = plotlyoffline(p)
% 要使用 plotly()在线绘图，可参考复合气泡散点图代码的最后部分
```

绘图结果如图 11-15 所示。

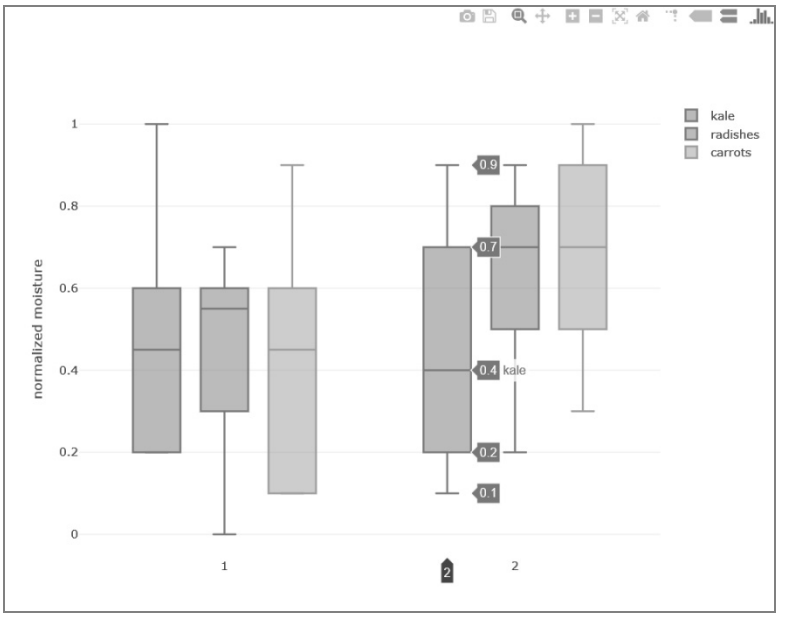

图 11-15　箱型图

5. 组合图

组合图就是将多种图形绘制在一张图中，其实前面已经展示了同时绘制多图的例子，只是绘制的是多幅同种图形，下面讲解简单组合图形的绘制方法。例如，柱形图与线形图组合的绘图代码如下。

```
    trace1 = struct(...
        'x', [0, 1, 2, 3, 4, 5], ...
        'y', [1.5, 1, 1.3, 0.7, 0.8, 0.9], ...
        'type', 'scatter');
    trace2 = struct(...
        'x', [0, 1, 2, 3, 4, 5], ...
        'y', [1, 0.5, 0.7, -1.2, 0.3, 0.4], ...
        'type', 'bar');
    data = {trace1, trace2};
    layout = struct( 'width', 800, 'height', 650); % width and height are
required parameters

    p = plotlyfig;
```

```
p.data = data;
p.layout = layout;
p.PlotOptions.FileName = 'bar-line';
plotlyfig.PlotOptions.FileOpt = 'overwrite';
response = plotlyoffline(p)
% 要使用 plotly()在线绘图，可参考复合气泡散点图代码的最后部分
```

在数据中，设置参数'type'为不同类型，即可同时绘制出不同类型的图形。

绘图结果如图 11-16 所示。

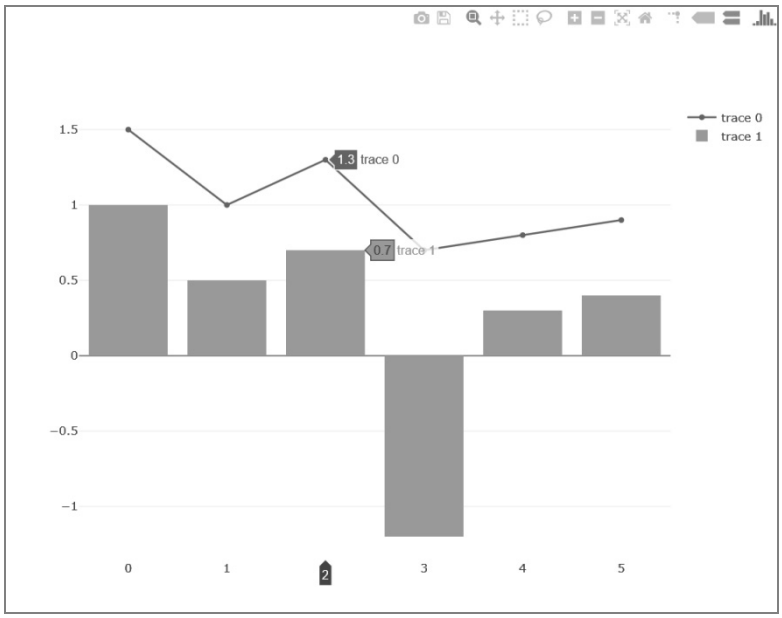

图 11-16　柱形图与线形图组合

11.2.4　更多扩展

除以上常用的经典用法外，Plotly 工具箱在 MATLAB 中还有很多扩展功能。

可以说，数据可视化在各个行业所需要的图，Plotly 都可以结合 MATLAB 本身的绘图功能加以实现，如建筑方面的 3D 图、物理学中的能量图、试验模拟用到的动图、信号处理中的实时更新变化图、金融经济数据常用的面板数据展示图、组合

展示的双坐标轴图和数据对比用的多子图等。

Plotly 作为最好的数据可视化工具之一，还有众多有吸引力的功能待我们去学习与应用，有兴趣的读者可以参考官网（https://plot.ly/matlab/）中的介绍和案例。

11.3　Plotly在JavaScript语言中的应用

Plotly 是一款独立的 Web 版数据可视化工具，界面友好，提供强大的互动性操作。Plotly 的绘图底层使用的是 plotly.js，plotly.js 基于 D3.js、stack.gl（WebGL 组件库，由 Plotly 团队的 Mikola Lysenko 领导开发）和 SVG，用 JavaScript 在网页上实现了类似 MATLAB 和 Python Matplotlib 的图形展示功能，支持 20 种图形，包括 2D 和 3D 图形，交互流畅，足以满足一般科学计算的需要。

本节主要介绍基于 JavaScript 接口的 Plotly 的使用，使用方法非常简单，只需要在独立的 HTML 文件中调用 plotly.js 绘图函数库就会生成基本的图形。

下面介绍最常用的几种图形格式：线形图（Line）、散点图（Scatter）、条形图（Bar）和扇形图（Pie）。

11.3.1　基础入门

Plotly 绘图分为两种情况，一种是离线绘图，另一种是在线绘图。

1. 离线绘图

笔者使用的是 winpython 开发的 Plotly（详见本书 1.2 节安装与安装环境），在 winpython 环境中（在其他环境下方法类似）使用 pip 命令安装 Plotly 模块。

```
pip install plotly
```

安装成功后可以在%\WinPython-2bit361\python-3.6.1\Lib\site-packages\plotly\package_data 目录下发现 plotly.min.js 文件。

首先，新建一个项目，命名为 plotlyweb，项目可以被理解为一个网站，在项目中新建文件夹 js，存放 Plotly 需要加载的 JavaScript 绘图函数库文件 plotly.min.js，创建好的项目目录结构如下。

```
plotlyweb
|
└──js
       |
       └──  plotly.min.js
```

然后，在 plotlyweb 项目下，新建 HTML 文件，命名为 pl1.html，在 HTML 文档中使用<script>标签加载 plotly.min.js 文件，代码如下。

```
<script src="js/plotly.min.js"></script>
```

接着，在 HTML 文档中创建一个空的 div，宽为 600px，高为 250px，用来绘制图形，代码如下。

```
<div id="tester" style="width:600px;height:250px;"></div>
```

最后，使用 Plotly.plot() 函数创建交互式 plotly.js 图表——一个简单的曲线图，代码如下。

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Javascript 与 Plotly</title>
  </head>
  <body>
    <div id="tester" style="width:600px;height:250px;"></div>
  </body>

      <script src="js/plotly.min.js"></script>
    <script>
      # 得到 HTML 页面中 id 为 tester 的 div 节点对象
      testerDiv = document.getElementById('tester');

      Plotly.plot(testerDiv , [{
         x: [1, 2, 3, 4, 5],
         y: [1, 2, 4, 8, 16] }], {
         margin: { t: 0 } } );
    </script>
```

```
</html>
```

在本案例中最重要的是设置 plot()函数的参数。

- 第一个参数：在 HTML 中需要渲染的图形使用的 div 节点对象。
- 第二个参数：设置 *X*、*Y* 轴的坐标数据和 margin（边距）的值。

使用浏览器打开 pl1.html 页面，稍等片刻就会渲染完成一个交互式的曲线图，得到的绘图结果如图 11-17 所示。

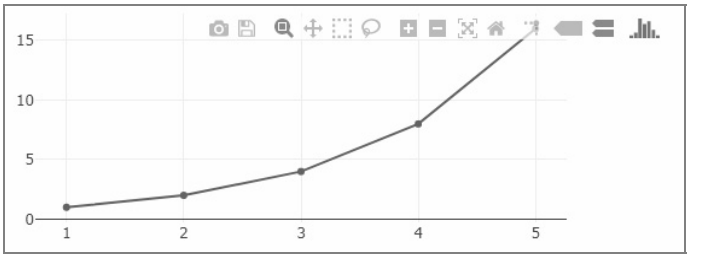

图 11-17　交互式的曲线图

2．在线绘图

除离线绘图外，也可以使用在线绘图，也就是该绘图不依赖本地的 plotly.min.js 文件。对于在线绘图，可以使用 Plotly 的 CDN 链接加载 plotly.js，提高 JS 脚本的加载速度。CDN 广泛采用各种缓存服务器，将这些缓存服务器分布到用户访问相对集中的地区或网络中，在用户访问网站时，利用全局负载技术将用户的访问指向距离最近的工作正常的缓存服务器上，由缓存服务器直接响应用户请求。

要把离线绘图方式修改为在线绘图方式，只要把 JS 脚本的加载方式修改一下就可以了。下面的代码是离线的加载方式。

```
<script src="js/plotly.min.js"></script>
```

替换为在线的 CDN 加载，代码如下。

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
```

修改后完整的代码如下。

```
<!doctype html>
<html>
  <head>
```

```
    <meta charset="utf-8" />
    <title>Javascript 与 Plotly</title>
  </head>
  <body>
    <div id="tester" style="width:600px;height:250px;"></div>
  </body>

    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <script>
        testerDiv= document.getElementById('tester');

        Plotly.plot(testerDiv , [{
            x: [1, 2, 3, 4, 5],
            y: [1, 2, 4, 8, 16] }], {
            margin: { t: 0 } } );
    </script>

</html>
```

使用浏览器打开 pl1.html 页面，绘图结果如图 11-18 所示，与离线的输出效果一样。为了提高 JavaScript 脚本的加载速度，在后面的案例中，我们都采用在线方式绘制基本的图表。

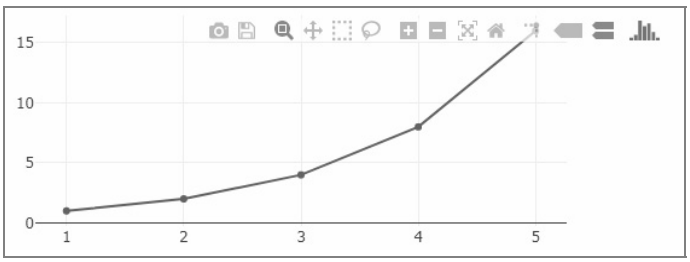

图 11-18　在线绘图结果

11.3.2　散点图

线形图又称曲线图，是最常用的图形类型。与传统的绘图软件不同，Plotly 没

有独立的线形图函数，而是将线形图（Line）与散点图（Scatter）全部用 newPlot 函数实现，且 newPlot 函数的 type 参数都需要设置为'scatter'，它们的区别在于 mode 参数不同。

本案例包括线形图与散点图两种图形的混合。新建一个 HTML 文件，命名为 pl_scatter.html，具体代码如图 11-19 所示。

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Javascript与Plotly</title>
  </head>
  <body>
    <div id="myDiv" style="width:600px;height:250px;"></div>
  </body>

    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <script>
    var trace0 = {
      x: [1, 2, 3, 4],
      y: [10, 15, 13, 17],
      mode:'markers', # 纯散点的绘图
      type: 'scatter'
    };

    var trace1 = {
      x: [2, 3, 4, 5],
      y: [16, 5, 11, 9],
      mode: 'lines',  # 纯曲线的绘图
      type: 'scatter'
    };

    var trace2 = {
      x: [1, 2, 3, 4],
      y: [12, 9, 15, 12],
      mode: 'lines+markers', # 散点+曲线的绘图
      type: 'scatter'
    };

    var data = [trace0, trace1, trace2];

    Plotly.newPlot('myDiv', data);
    </script>
</html>
```

图 11-19　散点图案例代码

使用浏览器打开 pl_scatter.html 页面，得到如图 11-20 所示的结果。

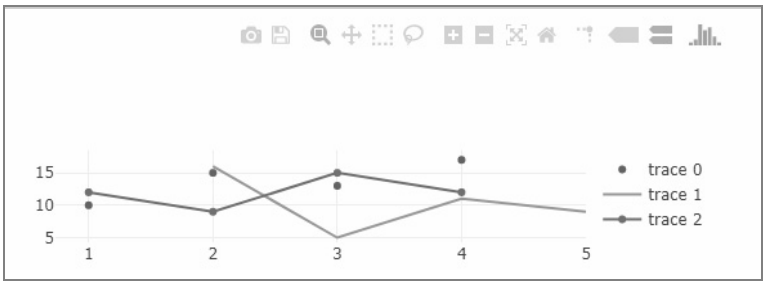

图 11-20　散点图

上述程序代码主要说明如下。

- 第 1 行，说明该 HTML 代码文件是符合 HTML 5 文档类型的。
- 代码第 3～6 行，HTML 的 head 标签。
- 代码第 8 行，在 HTML 文档中，创建一个空的 DIV 来绘制图形。
- 代码第 11 行，导入 plotly.js 的 CDN 链接。
- 代码第 13～18 行，设置散点图的数据源，这是一个 JSON 格式的字符串，trace0 对象的属性 *x*、*y* 对应图形在坐标系上的 *X*、*Y* 轴的值。trace0 对象的 mode 属性为 markers，表示图形为点图。trace0 对象的 type 属性为 scatter，表示渲染的图形类型是散点图。
- 代码第 20～25 行，设置散点图的数据源，trace1 对象的属性 *x*、*y* 对应图形在坐标系上的 *X*、*Y* 轴的值。trace1 对象的 mode 属性为 lines，表示图形为线形图。trace1 对象的 type 属性为 scatter，表示渲染的图形类型是散点图。
- 代码第 27～32 行，设置散点图的数据源，trace2 对象的属性 *x*、*y* 对应图形在坐标系上的 *X*、*Y* 轴的值。trace2 对象的 mode 属性为 lines+markers，表示图形为线形图+点图。trace2 对象的 type 属性为 scatter，表明渲染的图形类型是散点图。
- 代码第 34 行，声明一个数组变量 data，包括数据源 trace0、trace1 和 trace2。
- 代码第 36 行，使用 Plotly.newPlot()函数渲染散点图，第一个参数是在 HTML 中创建的空白 DIV 节点，第二个参数是加载的图形数据源。

查看案例源码可以看出，markers、lines、markers+lines 三个图输出的图表格式不同，是因为 newPlot 函数中的 mode 参数不同。

```
    mode = 'markers',# trace0 ,散点图
    mode = 'lines', # trace1,线形图
    mode = 'markers+lines',# trace2,线形图+散点图
```

11.3.3　条形图

条形图（Bar），也叫柱状图，是以长方形的长度为变量的统计视图，用来比较多个项目分类的数据大小，通常用于较小的数据集分析。例如，不同季度的销量、不同国家的人口等。

用 Plotly 绘制条形图与绘制线形图类似，不同之处是需要在 newPlot()函数中设置图形数据源的 type 参数为 bar。本案例新建一个 HTML 文件，命名为 pl_bar.html，具体代码如下。

```
<!doctype html>
<html>

    <head>
        <meta charset="utf-8" />
        <title>Javascript 与 Plotly</title>
    </head>

    <body>
        <div id="myDiv" style="width:600px;height:550px;"></div>
    </body>

    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <script>

        // 条形图的数据源
        var data = [{
            x: ['apple', 'orange', 'banana'],
            y: [20, 11, 25],
            type: 'bar'
        }];

        Plotly.newPlot('myDiv', data);
    </script>

</html>
```

使用浏览器打开 pl_bar.html 页面，得到如图 11-21 所示的结果。

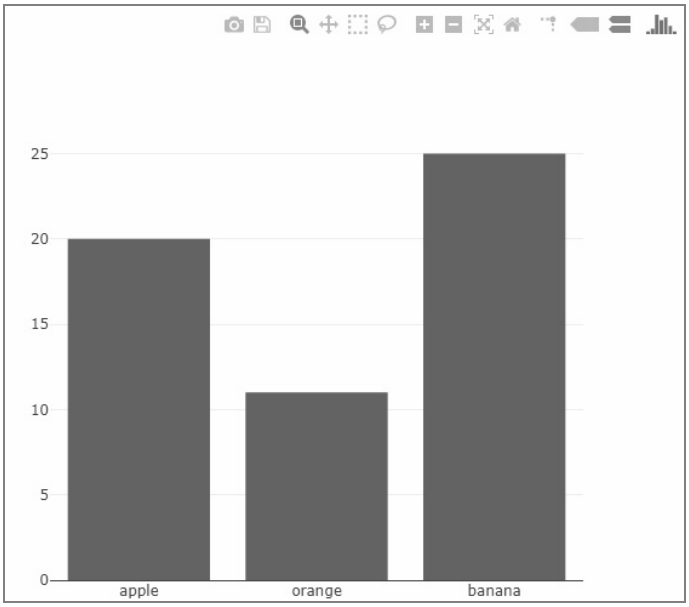

图 11-21　条形图

11.3.4　扇形图

扇形图（Pie），也叫饼图，显示一个数据系列中各项的大小与各项总和的比例。饼图中的数据显示整个饼图的百分比，如移动手机品牌占市场份额图。

用 Plotly 画一个基本的扇形图，需要调用 newPlot()函数，传递数据并设置样式。通过 layout 参数绘制出不同类型的扇形图。本案例新建一个 HTML 文件，命名为 pl_pie.html，具体代码如下。

```
<!doctype html>
<html>

    <head>
        <meta charset="utf-8" />
        <title>Javascript 与 Plotly</title>
    </head>
```

```
    <body>
        <div id="myDiv" style="width:600px;height:550px;"></div>
    </body>

    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <script>
        // data
        var data = [{
            values: [19, 26, 55],
            labels: ['A', 'B', 'C'],
            type: 'pie'
        }];

            // layout
        var layout = {
            height: 400,
            width: 500
        };

        Plotly.newPlot('myDiv', data, layout);

    </script>

</html>
```

使用浏览器打开 pl_pie.html 页面，得到如图 11-22 所示的结果。

在本案例中，核心代码是设置 layout 参数，设置扇形图的高度（height）和宽度（width），代码如下。

```
var layout = {
          height: 400,
          width: 500
        };
```

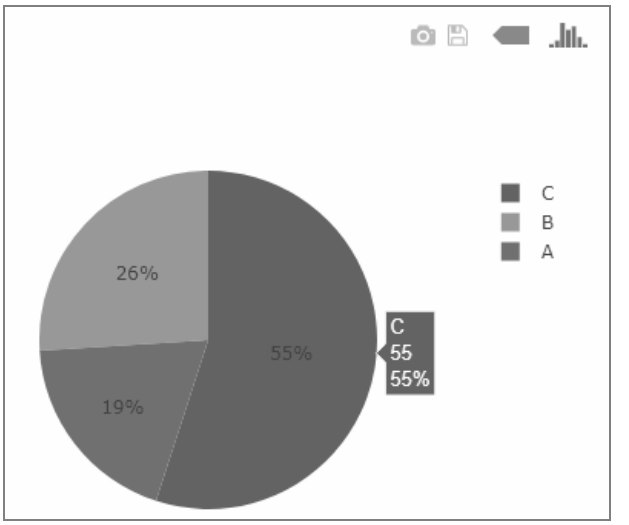

图 11-22　扇形图

11.3.5　更多扩展

Plotly 绘图是基于 JavaScript 技术而来的，所以对 JavaScript 语言天生支持，本节内容仅展示了 JavaScript 调用 Plotly 最基础的案例，对于更多的案例，读者可以参考 Plotly 的 JavaScript 官方网站（https://plot.ly/javascript/），在这个网站中可以找到自己所需的绝大部分绘图功能。